Title:	Modular composition environment: A tool for improvisation of conventional electronic music.
Author:	Joaquín Aldunate Infante
Thesis advisor:	Koray Tahiroğlu
Thesis supervisors:	Teemu Leinonen, Markku Reunanen
Submission:	October 2018, Espoo, Finland
Program:	Masters in New Media Design and Production, Aalto University

1 Table of contents

2 Abstract

Keywords: Electronic music, Live, improvisation, product design, affordance, divergence, divergency, sequencer, nime, controller, modular, environment, music composition, hardware, new media.

This production thesis sets out to create a tool for live improvisation of music that allows musicians to create and modulate musical patterns in real-time and reduces the need for pre-recorded or pre-sequenced material. It starts by defining the scope of *conventional electronic music* and then explores the shortcomings of current tools in relation to the divergency of music making.

The project is based on the author's previous experiences in the live improvisation of conventional electronic music, and thus it starts by surveying the currently existing tools. After that, it focuses on the iterative design process of modular environment, taking the modular synthesizer as a conceptual starting point. These processes led to the development of composition devices which are expressed through a hardware user interface, in a modular environment.

This project finds that the shortcomings in divergency of current music improvisation tools come from the fact that musical modulations in an improvisation tool are inherently limited by the available procedures of any given system. While composition tools such as modular synthesizers lack this limitation they do not have the discrete musical abstractions required for conventional electronic music. The production project thus focuses on the design of a modular environment that could permit re-purposing of procedures that process discrete musical events. The outcome of this project is a new performance environment that can be used to generate more diverse improvisations of conventional electronic music.

3 Introduction

In this chapter, the whole context of the project is explained, starting from the personal motivation of the author. After this, a theoretical framework is established: the basic concepts are explained, so that it is possible to establish the intended meanings of the words being used during the project. This leads to many distinctions that help focus better the scope of this project into a very specific domain. After this, the current state of the art is analysed by showing an overall map of current ways that electronic music is performed live, leading to the discovery of the gap which this project intends to solve or explore. The introduction chapter ends with the statement of this gap, and how it can be interpreted in terms as stated in the theoretical framework.

3.1 Motivation

For some years, I have been developing the ability to stage live performances of electronic dance music using various tools. These tools have served well; however, a feeling of being limited by the tool has always been more notorious than my own impression of being able to do something new with it. I would buy a tool expecting that it would help me do something in my live performances, but there was always the same problem. This forced to adapt my performances to the ways the tools worked, while I was expecting it to be the opposite. Using performance tools was disenchanting. (I have to admit, however, these imposed ways of playing also taught me most of what I know about performing live.) At the beginning, I set out to create a music-making tool of my own that I could use in my live performances and to customize its behaviour in such a way that I could perform improvised musical modulations that would be otherwise impossible. The thesis work took me onto a slightly different, more interesting path.

3.2 Theoretical framework

It is important to establish what it is being said when using certain words. In highly-specific subjects such as this, it may happen that a reader comes with different definitions of certain concepts. In order to be able to use these words, we need first to establish which of all the possible meanings of that word is going to be used in this thesis. In addition to this, it is important to delimit an area of work when speaking, for example, about electronic music. This is why in this theoretical framework some remarks are added in order to distinguish a specific domain of electronic music performances among the vast area which such concept encompasses.

3.2.1 Affordance

In order to initiate a discussion about possibilities of musical devices, it is necessary to introduce the widely known concept of affordance. This concept is credited to Gibson, and it characterizes the relation between an organism and its environment (You and Chen 2007). In the words of Gibson, "[t]he affordances of the environment are what it *offers* the animal, what it provides or furnishes, either for good or ill." (Gibson 1979, 127) From the perspective of design, each object may or may not afford different uses or relations to a user[1]. Although the affordance of, say, a chair includes sitting on it for a human, this

1 note that Gibson's notion of affordance focuses on the relationship between any animal with its environment. Within a design process, the animal to be considered is most likely to be a

relationship may extend beyond the initial design intention of the object, such as standing over or throwing it.

The application of affordance to the design of complex instrumentation puts this term into crisis, because according to You & Chen, affordance is limited to what can be perceived without effort (You and Chen 2007, 25). This delimitation derives from Gibson's later expansion in relation to perceptual processes. In these terms, the concept is useful for the design of physical products because it provides a clear way to evaluate how easy it is to understand a product. Within the topic of development of musical instrumentation, however, it will be necessary to ignore the latter distinction. This is because, although the affordance of most instruments is clear[2], using the instruments musically do require further mental effort than what is directly perceived. Let us take a Kaoss Pad as an example: once it is turned on, all the interaction possibilities are clear. The touch-pad displays moving lights which intuitively suggests touching, the encoder also indicates that it can be rotated. The buttons are also clearly push-able. How to use this tool musically, however, needs further reflection: a musician needs to know what to plug into the unit's input and output terminals. The user also needs to be aware of the desired BPM at which to run the unit. In order to use the unit effectively for its function, it is necessary to go beyond what the affordance shows. In a broader sense of the concept, however, it supports interesting views to analyse a device in relation to higher-level actions such as *composing* or *looping*. The term affordance, therefore, will not be relegated to what is intuitive, but will also include what an object facilitates regardless of how much it needs reflection or knowledge.

3.2.2 Linear and divergent thinking in music

Divergent

> a: moving or extending in different directions from a common point ("Merriam-Webster Dictionary, Definition of Divergent" 2018)

In the field of psychology, divergent thinking is associated with creativity in many studies. These works help build a richer idea of creativity. The idea of divergent thinking can be credited to Guilford (Runco 2011, 400). His intention was to highlight the relevance of creativity as an exertion of intelligence (Guilford 1970). Among many other types of creativity, he identified creative activities whose intended outcome is largest possible quantity of solutions as "divergent production" (Guilford 1970, 159). Guilford's work appears as the main guiding principle for a concrete definition of divergent thinking in Runco's entry in the encyclopedia of creativity (Runco 2011).

---

human, and the most likely factor of the environment is the object in question.

2 some examples: buttons are clearly push-able, decks are clearly spin-able.

Guilford formed three indicators for divergent creativity: fluency, originality and flexibility. Fluency represents the number of ideas provided by the test subject. Originality represents the infrequency of such ideas in comparison to the other test subjects, and flexibility represents the conceptual difference among the ideas given by the same subject (Runco 2011, 401). For easier reference, Fig. 1 provides a graphical representation of these variables, being added one by one. Fluency appears in the figure as the number of ideas, not needing these to be varied. Flexibility appears as a varied and flexible group of not necessarily original ideas. Finally, originality appears in Fig. 1 representing the application of the three factors. Note that the figure is only for reference and not an accurate depiction of the three variables: originality can only be assessed across different test subjects. If the intention is to create a tool that allows a more divergent musical expression, these three key aspects of divergent thinking form a valuable design focus.

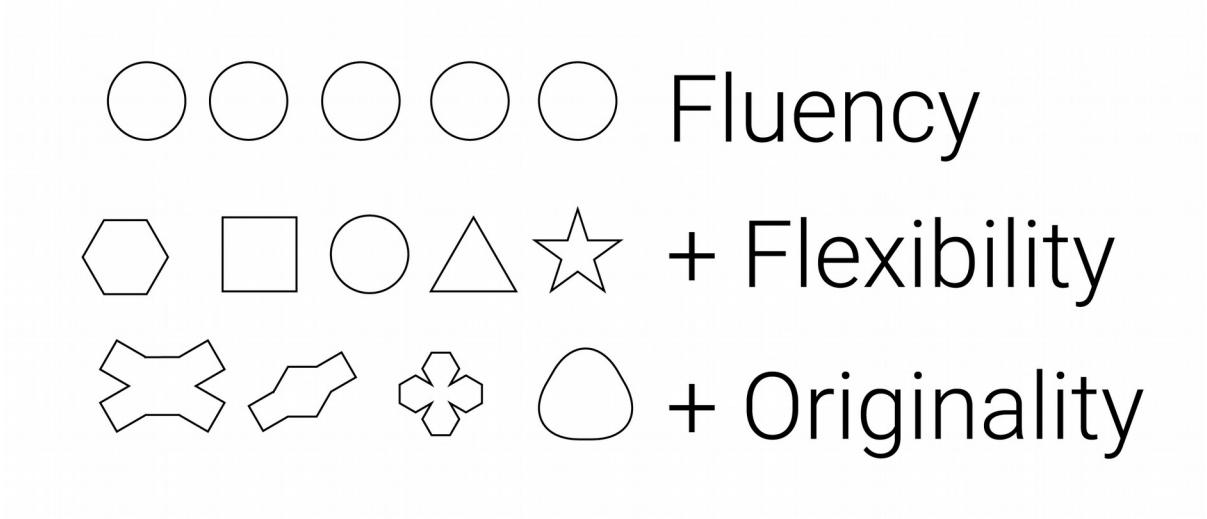


Figure 1: Representations of flexibility, fluency and originality

The *fluency, originality and flexibility* definition for divergence is easily transposed to the domain of music making. The Brocs thesis work sought an idea of divergence in terms of musical outcome from a less informed perspective (Aldunate Infante 2013b), yet bringing an interesting idea to this discussion. As exemplified in Fig. 2, listening appears as the least musically divergent activity, since the musical outcome cannot be altered beyond subjective perception (e.g., focusing on an instrument, liking or disliking). Composition, in comparison, is a more divergent activity, since it consists on creating new musical pieces that did not previously exist (Aldunate Infante 2013b). Any musical activity could be theoretically assigned to a range along this divergence axis, leading to the idea that each musical practice possess an inherent level of potential for divergence. In other words, each musical activity or musical instrument *affords* different levels of divergence. This affordance, or divergence-potential which is inherent in an activity hereafter will be termed as *divergency*. The distinction being made, is between a divergence that is the responsibility of the performer of the activity, and the derived concept of *divergency* which is facilitated by the activity being performed. The first notion, being based on the subject, is a study subject of psychology. The focus on divergency, however, is a subject of design.

This project will focus on this divergency, as the interest is not to improve personal improvisation skills, but to produce a product which affords divergent improvisation.

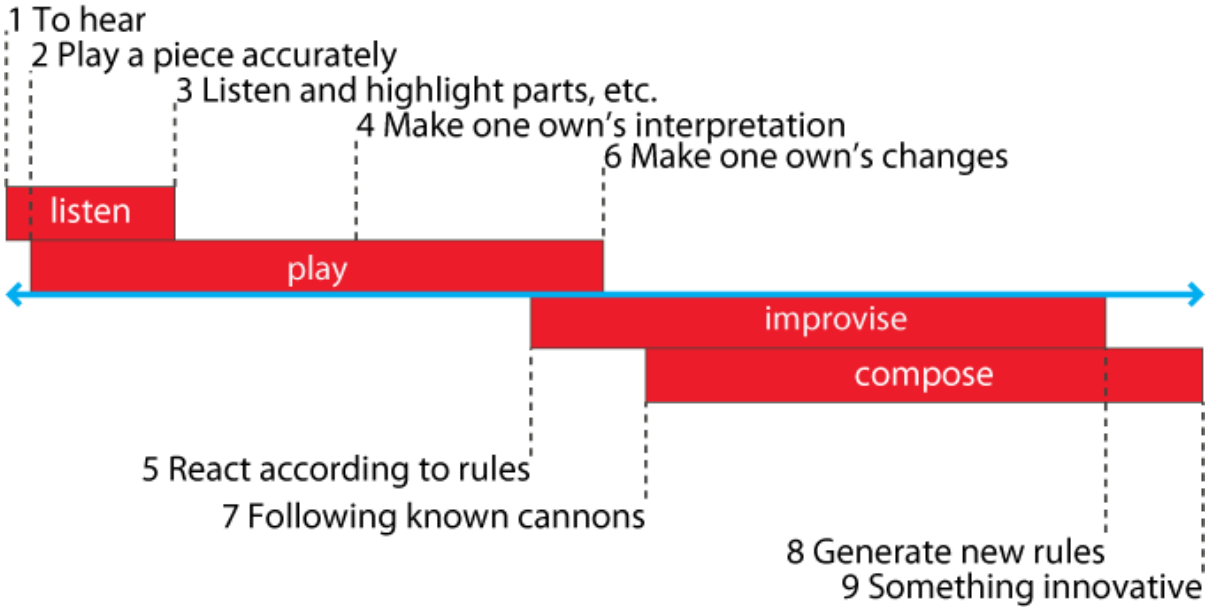


Figure 2: Linear-divergent spectrum of musical activities, translated from Aldunate Infante (2013b)

Non-divergent activities are different when considering the domain of psychology versus the domain of musical activities, or music making tools. In psychology, there are many other intelligence activities which are not classified as divergent, according to Guilford (1970). The interest of this project, however lies exclusively in the divergent production rather than other activities such as convergent production or memorization. In the terms that were defined above, divergency is defined as a single axis variable that ranges from a narrow to a wide range of possible outcomes. For term divergency, linearity will be used as the opposite term, to express that activities of less divergency have a narrower scope of possible outcomes.

This thesis will be focusing in the ability to be divergent specifically in live performances and/or live improvisations. The divergency of a musical tool really is a variable of potential divergence, since once a musical piece takes place in a performance, all the other possible musical pieces do not. Divergency, the potential for divergence, therefore, consists on the created piece plus how many other pieces of music are not being created, but are possible. Non-realtime musical activities, such as composing, have a vast divergency. This is because the composer has the opportunity to invest a time that is longer than duration of the piece, whereas a live performer only has the duration of the piece as the available time to produce it. Whereas a composer of pieces has the possibility to go back in time to alter the piece in any way, a live performer can only alter the present, and with certain tools, the future of the piece. For these reasons, there is little use in the design of tools for composition, but there is a need to design tools for live performance or improvisation when it comes to the divergence of the musical expression.

3.2.3 Different cultures around live electronic music

Divergency is not a desirable value in every context. For instance, a non-divergent practice such as singing known songs, or learning to play composed pieces are highly valued activities. Music as a collective experience can be enjoyed in a *cover band* concert or a dance party where pre-recorded music is played with minimum alteration. Uniqueness and live-ness[3], however, is appreciated in some electronic-music related social contexts, as it will be discussed in the following section. All this amounts to determining a social scope where a tool for divergent music making is valued but currently limited.

Togetherness is a concept that is tightly related to the subcultures of electronic music, around which there is an on-going discussion. Sociologically, this concept is also termed as "solidarity" (Kavanaugh and Anderson 2008). The term describes how, at electronic music parties, all the participants feel like being an integral part of the group of people and flow of the party, participating in solidarity. The discussion is about whether this feeling of togetherness in dance music is a product of the underground history of electronic music (Straw 1993), an effect of the inherent characteristics of clubbing and the music (Reynolds 1999; Butler 2006, 72), or an effect of the use of drugs (Kavanaugh and Anderson 2008). Anthropologists like Kavanaugh and Anderson propose other sources for the social bond: among other reasons, there is collective dancing, staying up late at night in groups, and collaborating in the organization of events (Kavanaugh and Anderson 2008, 191). Arguably this phenomenon is caused by the combination of all these elements. The fact however, where all the authors seem to agree, is the existence of this collective aspect in the experience of electronic music parties. This collective aspect is relevant to the discussed topic, as it will be discussed hereafter.

To the reader, it might appear that the formal characteristics of electronic music have little to do with the emergence of solidarity. Electronic music, nonetheless, does have inherent characteristics that foster audience-performer interaction or solidarity. James Andean & Alejandro Olarte in *Sound, Music and Motion* work, assert that musical predictability and danceability are related (Andean and Olarte 2012, 2). This idea is easy to accept, since predictability is not exclusive to electronic music, and many other danceable music styles across history possess some recurrent patterns (e.g., Foxtrots, Cumbia, Waltz, Salsa). This gives a reason why collective parties have always taken place around repetitive music of predictable patterns: predictability permits the dancing participants to know with certainty what are the upcoming musical events. Some styles exhibit a more complex set of rules that requires study on behalf of the performers (e.g. Flamenco), but which again, are aimed to make the future musical events predictable. Apart from allowing dancers to synchronize their movements with the music, it also enables coordination among dancers, facilitating synchronicity among the participants. This synchronicity between audience and musician integrates the audience into the musical process, leading to the idea of participating all together in a collective event. In relation to laptop and IDM performers, Emmerson (2007) states that listeners can also become an integral part of the pieces

3 The quality of music being produced live, in the stage.

themselves, which in many cases are intended to be a 'symbiotic' (Emmerson 2007) composition of the performer with the audience.

Another inherent characteristic of electronic music that encourages a sense of participation is the intertextuality and collective production. Electronic music is created from borrowed material, which implies a rich *intertwining* of content, often expressed as *intertextuality*. Although intertextuality is very common in western classical music (Vasquez 2016), "[t]he use of the sampler has made this intertextuality more apparent, since a song can be created from the sequencing of snippets of sound as well as from recognizable fragments from other records." (Rietveld 1995, 2) This intertextuality is a natural consequence of the use of recorded material as an instrument. The practice started with tape reels by concrete musicians (Warner 2017, 17–55), notably Mauricio Kagel's *Ludwig Van* which could be considered as the first remix ever practised (Vasquez 2016, 17). These derived into practices such as deejaying and sampling (Warner 2017, 89–169) in some cases still recurring to tape-related techniques (Kirn 2011, 38, 46). Many cases of current NIME research also search for augmented collaboration features. Two examples of this are the *PESI Extended System* (Tahiroğlu, Correia, and Espada 2013; Parkinson and Tahiroğlu 2013) and *Reactable* (Kaltenbrunner et al. 2006). It is very clear, thus, that the concept of collaboration is present at the roots of electronic music.

In addition to sampling, as Will Lynch explains; "it's[sic] normal for artists to pay other artists to execute their ideas in the studio, then downplay their involvement later on, sometimes not crediting them at all. As a result, many artists get credit for more work than they've[sic] done, or are even capable of doing. The average listener is none the wiser." (Lynch 2017) This underlines that music production can be a participatory practice. The role of the author in this context can be anywhere between a composer and a mere connector of other actors. This further proves that electronic music creation is a collective process, which further emphasizes the notion of the genre's association with solidarity.

Dancing at parties of electronic music can either be characterized as collective and participatory or as individual. If we compare a *disco* party to one of electronic music, it will be noted that instead of finding dancing couples, people are found dancing on their own (Butler 2006, 36), facing the deejay or performer. This can either be read as each participant having an individual experience with the music or, to the contrary, as every participant taking part in a collective experience. Malbon, highlighting the social role of music parties, suggests that an audience which knows how to listen is an essential part of any music performance. The absence of such audience, according to Malbon, renders the performance "useless." (Malbon 2002, 82) Hilegonda Rietveld, as cited by Butler (2006), underlines this further: "[i]t is only when played to and interacted with a dancing crowd, that house music, as a medium, is complete." (Butler 2006, 13) Additionally, many guides for deejaying, when not focusing in the technical part, will explain that a good deejay will select its tracks according to the present audience (Walsh 2018). All these assertions suggest that the experience of dancing in an electronic music party is more a collective experience than a multiplicity of isolated experiences.

Additionally, the scarce use of lyrics, has caused the discourse of the genre to be undefined, thus lending itself to a heterogeneous group of people.

> *The crowd is unusually diverse as well. Teenagers from downtown Detroit mingle with suburban kids from across the Midwest. A young raver in a wheelchair, her arms covered from wrist to shoulder with plastic beads, spins about near a group of gay men. A middle-aged African-American woman in a jogging suit listens intently to the music, her eyes closed, while a tour group from Amsterdam takes in the scene. People of all stripes, from all walks of life, have come here to hear this music, yet they respond as a group. The beat can not only be heard, it can be seen in their movements, and felt in their bodies.* (Butler 2006)

A hip-hop song, with lyrics, must sing about something, and will portray a political or moral stand, which an audience may sympathize with or disdain. The case is the same with with pop-stars, whose aesthetics build a very strong image of a particular social ethnology. Electronic music, however, seems to offer a broader field for different sets of personal values.

One exception for this openness which needs highlighting, is a certain level of male sexism in the sub-culture. This can be seen in way that the role of female deejays is depicted as special or non-normal (Rietveld 2013, 8) and in the fetishized representations of women that are portrayed in the scarce times there are lyrics present. In addition to this, as Denise Dalphond explains in an interview that was documented by Peter Kirn, electronic music record stores are very male-centred and discourage the interest of women (Kirn 2011, 43). The same is the case when it comes to the role of women's musical interest in general: "In my experience, men either assume you don't[sic] know anything, or think that your interest in music is hot and turn it into a sexual thing" (Kirn 2011, 43).

The term EDM is a contraction of *electronic dance music* and is often used to include this whole genre, one example being "unlocking the groove" (Butler 2006). Under the term EDM, there are different notions of live performance which, for the purpose of this project, need to be distinguished. In a context such as deejaying, where the performance material are pre-recorded music tracks, the predominance of the author inverse to the one of the performer. According to Straw (1993), in the mid 1970s deejays started concealing the identity of the tracks being played for the party. Straw claims that the "credibility of dance music's professional culture have been built upon an investment in secrecy" (Straw 1993). The credibility of the deejay is related to the non-disclosure of the tracks. This intention to conceal might not appear as evident. In some cases the intention of a performance is not the concealment of the tracks, on the contrary, intentionally letting the audience recognize what is being played. In the case of the deejay this idea would seem to make the author role less prominent. In the case of live electronic music shows, or deejays who play productions of their own, authorship remains prominent regardless of how recognizable the (instantiation of the) tracks are. Whichever the case, be deejays who conceal their track listing or live musicians which play their own compositions, authorship is a desired feature of live shows.

Different accounts of electronic music history disagree about the value of a musical piece being recognizable. Whereas in some accounts, such as Peter Kirn's (2011), the popularity of a certain musical piece is part of a positive feedback loop in popularity. In contrast, the

appreciation for *white labels* seem to prove the opposite true (Hesmondhalgh 1998; Straw 1993). Some EDM cultures seem to appreciate the familiarity of the track, and expect deejay sets which are composed mostly of known recordings. Some other sub-cultures, by contrast, expect the performance to be familiar in only style, but value its uniqueness.

This difference is relevant when it comes to whether a musical performance intends to be divergent or not. Where some live shows tend towards a recognizable reproduction of the pieces (since the value is popularity), other performances seek to be unique and unrepeatable (because the value is uniqueness). This thesis, therefore, is focused mostly on performances which seek uniqueness: this is where the idea of improvisation makes the most sense. It is also possible that performances of recognizable pieces may benefit from a platform for improvisation to attain previously unseen versions over those pieces. In both cases, the author as well as the performer emerge as prominent figures in the performance.

3.2.4 Differentiation between experimental and conventional music

In a widely known comedic episode, Karlheinz Stockhausen listened and criticized some tracks of more popular electronic musicians such as Aphex Twin. According to this story, Stockhausen wrote about Aphex Twin's compositions:

> "*I think it would be very helpful if he listens to my work Song of the Youth, which is electronic music, and a young boy's voice singing with himself. Because he would then immediately stop with all these post-African repetitions, and he would look for changing tempi and changing rhythms, and he would not allow to repeat any rhythm if it were [not] varied to some extent and if it did not have a direction in its sequence of variations*" (Witts and Stockhausen 1995, 32)

This story demarks a clear difference between the worlds of experimental and conventional electronic music. Conventional music tries to satisfy the need for rhythm and the, perhaps, hedonistic lust for melodies and harmonies composed according to a western canon. In this thesis, conventional music is distinct from this notion of experimental music which seeks to disrupt, or *work without* conventional notions of music, such as rhythm, harmony or melody. From this point on, the term *conventional electronic music* will be used as a subset of electronic music. Where in electronic music there is space for sound performances, the limits of conventional electronic music are demarcated by the use of conventional musical abstractions, such as meter, rhythms, patterns, loops, tones and scales. This delimitation somehow connects an ambit of electronic music to the previously developed tradition of classical music, rock, jazz, and so on.

This thesis project will focus on the more conventional styles of electronic music. For experimental electronic music and experimental music in general the tools seem to be inherently sufficient. As explained, since experimental music does not constrain itself with

conventional rules, it allows the use of any object as musical artefact, and the object's affordances can represent the rules of the piece. In this sense, there is no use in the invention of tools with enhanced divergency in experimental music making since the selection of the tool forms part of the musical composition process (Maraš 2011). In a case where this would not be true, it would be necessary that the musical artefact is created by the artist themselves, since the resulting musical piece, expectedly, would be bound by original rules. In an experimental music process, henceforth there would be no use for an electronic music tool, unless it is used in ways for which it is not intended.

3.2.5 The concept of music solo act in electronic music

It is said that the appearance of the tape recording technology had an impact of similar magnitude than the impact photography had to painting (Warner 2017, 17). The capacity to record sounds created a philosophical instability around sound and music, creating a whole new field of research and exploration. As suggested by Daniel Warner, the appearance of a practical possibility can have an impact on matters such as the meaning of natural phenomena. When there is the ability to record, sound events can be re-contextualized in new ways (Warner 2017, Chapter 1). Recording techniques facilitated sonic productions by individual artists, as for example, Pierre Schaeffer. The ideas behind early *musique concrete* explorations with recorded material, started gaining acceptance by wider audiences while recording technologies infiltrated popular music genres. The tape reels became standard music studio equipment, normalizing the use of post-production. Consequently, music that is composed on the basis of techniques rather than instrumental performance started emerging organically. In the 80s, with the many developments around digital instrumentation, it became possible to record, alter and play sampled sounds at live stages.

While the role of a musician prior to recording technologies was crucial to the existence of any music (because mechanical instruments do not play themselves), after recording or rendering it is possible for music to exist without there being a performer. Given this, a space was born for performances using music records, giving birth to the idea of a deejay. This use makes enables a single person to facilitate the live presence of complete musical pieces without needing a band.

Among other techniques, a live music performance can be performed by a single person thanks to looping. A loop is a musical fragment (sampled or composed) which has a length in relation to a musical meter. For example, a musician could be working with two loops: one which is a four beats long bass melody, and another which is a sixteen beats long sample of a drum pattern. A single musician can launch, stop and edit these loops, which keep repeating. With an adequate user interface, it is possible for a musician to perform in real-time a polyphonic piece.

These factors explain why most electronic music is played by a single live performer, and why many tools are oriented towards allowing solo performances. Looping alone,

however, has not been enough. Looping tools now a days also integrate with options to produce modulations to these loops. These modulations can either take place in terms of acoustics (e.g., signal processing effects, re-sampling) or in terms of composition (e.g., duplicate a sequence, shift an octave, re-order a pattern). This is particularly true with tools such as Ableton Push or Maschine, where the machine allows multitimbral compositions; all to be managed from a single user interface node. In this way, a variety of loops can be produced from one source loop. The same has been true in the case of deejaying, which is, in the vast majority of the cases, performed individually.

3.3 Musical devices and their performance paradigms.

Thus far, divergency was defined in such way that allows a particular description of musical activities. This thesis has also asserted that a music-making tool which affords divergent improvisation could be appreciated in a certain musical ambit or social context. The last premise that needs clarification in order to support the thesis, is related to current tools and their divergency in live improvisation of conventional electronic music. The following chapter explores the different ways that divergency is attempted in live musical performances, what their limitations and what their advantages. In other words, the process to follow defines the state of the art in live electronic music improvisation.

In order to understand the broader context of a performance tool, different musical instruments were surveyed to understand different approaches toward live performance interaction. This provided both, with an overview of performance possibilities, and with a categorization of performance paradigms. The categorization was not performed by assigning instruments according to formal characteristics (such as shape, size, presence of buttons). That categorization, however useful for some purposes, would not provide a notion of the variety of techniques to perform music, but with an atomized gamut of similarities and differences among items. Furthermore, a categorization of *different ways to perform music* is likely to have unclear boundaries, (Frey, Gelhausen, and Saake 2011) defeating the purpose of this type of categorization. A method was used, instead which led to a categorization more similar to *prototypes* or *exemplars* (Frey, Gelhausen, and Saake 2011) where items need not to be perfectly matching elements of a category, but be related in such way that reflects that there are many features in common. In this way, rather than attaining a categorization of elements, each performance type obtains neighbourness to others.

Each of the surveyed instruments was taken as a proxy of their intended performative use.[4] These performative uses can be found in documentations of different performers using these tools for musical performances. Their use can also be inferred from online

4 e.g., a piano is intended to be used by pressing the keys albeit some musicians could use it in other unexpected ways such as touching the strings or burning it down.

documentation and manuals. For each of the items considered, a reference was added where it is possible to review their intended use, which can be seen in the appendix. The relations between instruments was explored by relating the found elements one to each other. These relationships are established by intuition. This step is expressed by the links among elements in Fig. 3. By following this method, the group of surveyed techniques tend to form groups or neighbourhoods, which help discretize the conceptual approaches toward musical performance. The bigger, blue captions seen in Fig. 3 give name to these groups, and each element relates to each group to different extents.

A broad scope of electronic music performance instruments was selected to compare and make groups, but the different nature of many of them posed some challenges. Korg products are usually unique and with a delimited functionality, such as the Kaossillator. This makes these products easy to place into a category. Some groups of products, however, needed to be considered as a single item, while other single products needed to represent a whole category of similar products. One example is the category of deejay controllers and decks: there is a broad variety of products, each with some differences. For this case, the CDJ[5], was taken as the main example in representation for deejay consoles in general. The piano, although not being an electronic music instrument, it stands as a reference point between classical instruments and the ones being analysed, as well as a proxy for mechanical instruments.

Another caveat to this grouping process is the variety of different relations between controllers and software. For instance, a performance using an Akai APC40 is very different than one that uses an Ableton push, despite that in both cases, Ableton is the intended host application. Should the controller be taken as a mere access point to the host application, both would be considered to be the same. There were important differences between how the performance operates depending on the controller, however, that needed to be taken into consideration. The decision in regard to this, was to consider the controller as the independent user interface, as if the host application would have been integrated in it, and it was not accessible via another user interface.

As a last remark in relation to this process, it was necessary to consider modular environments as a singular product. The most important case that reflects this, was Euro-rack. Although Euro-rack is not a product but a standard where different products can correlate, the system offers many modules which do not work on their own, but as part of greater systems, and it is also possible to find Euro-rack modules that could fit in any group. In addition, there are self-contained modular systems such as Reaktor or Reactable which could be used to build any other product, and hence, fit in any group. In this case, if the different modules of the Euro-rack environments were considered as singular items, it would make it necessary to consider virtual modules from Reaktor or Reactable as separate items as well. Euro-rack and other modular musical tools offer different user interface propositions more as a system than as individual units.

5 A deejay deck and controller created by the brand Pioneer.

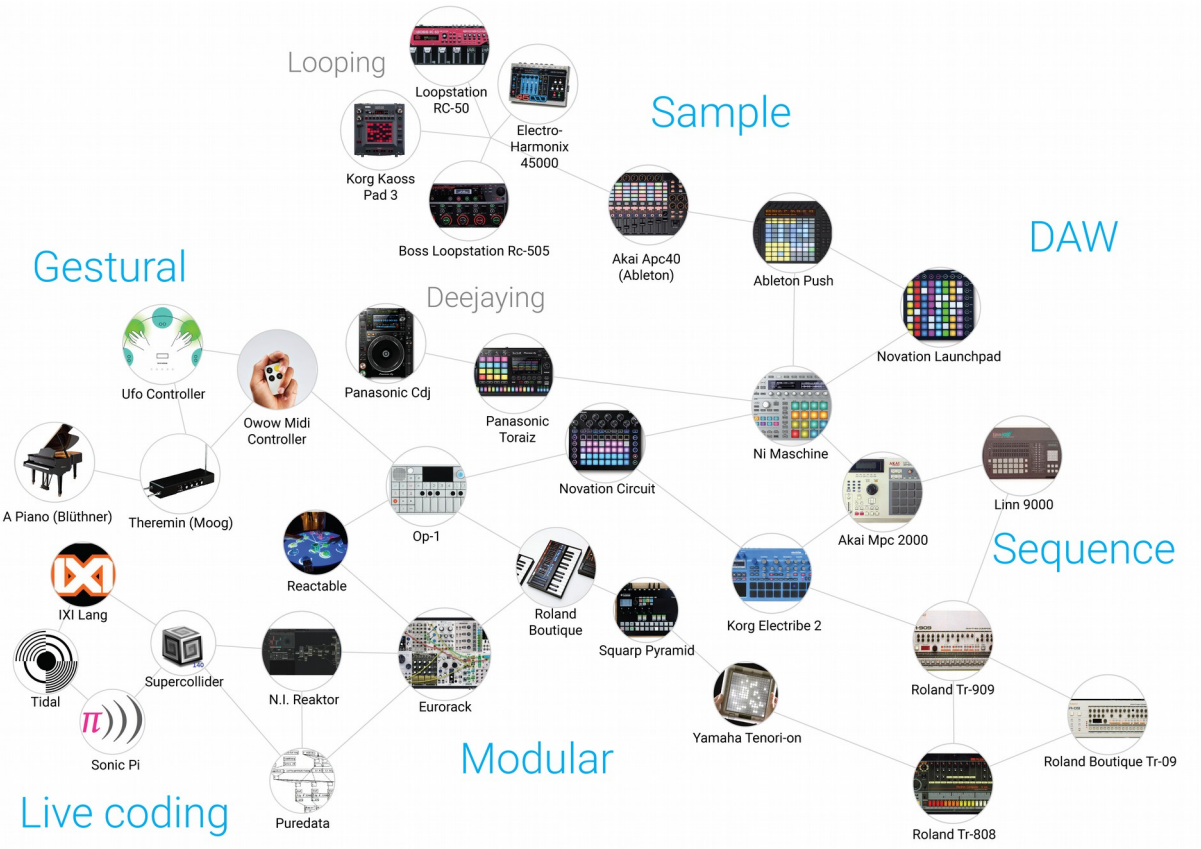


Figure 3: Intuitive grouping of music making tools according to how they are performed

3.3.1 Gestural-mapping based tools

The paradigm of gestural mapping is the most intuitive approach to design and understand electronic music instruments, since it mimics the relations humans have with mechanical instruments, while taking advantage of the augmented features that electronic instrumentation offers. One of the earliest electronic examples of this is the Theremin, where the distances between the performer's two hands and two antennas, would determine pitch and volume respectively. The logic behind gestural mapping is that having a real-time sonic response from a body action conveys the most intuitive interface for composition, which is exactly the same as with mechanical instruments.

Three current examples of this paradigm in a controller, are the Owow MIDI controllers (White 2018), AHNE (Niinimäki and Tahiroğlu 2012) and Tommi Koskinen's UFO controller (Koskinen 2015). Their main features are simplicity and granularity, taking advantage of the "decoupling" (Koskinen 2015) of the sound production from the action (Koskinen 2015, 9). One similarity among these two controllers, is that they offer specific mappings of a gesture to a musical parameter or event, assuming that these will be combined with other expression interfaces. A less obvious decoupling shown in these devices is between the user interface parameters and their association to a technical aspect of sound. The unnamed parameter approach, opposes to the presentation of parameters in

traditional synthesizers, where gestures (most likely knobs) are labelled with signal processing terms such as low-pass or pulse-width. Some of these gestural mapping tools try to encourage intuitive use by removing the names of the parameters so that the users rely more on their audition than sound synthesis related concepts.

Most of the high-end synthesizers work under gestural paradigm, given that their design focus is on the sound design. The parameter controls are relegated to the commonly used panel with knobs and keys. In many cases, it is assumed that more advanced sequencing will be provided by a sequencer using a control input (e.g., MIDI, CV, OSC). One extreme example of this are the Roland Boutique synthesizers, which do not possess their own keyboards, becoming, in a sense, modular.

The use of gestures as expressive input for musical performance offers a broad spectrum of possibilities. For multitimbral composition, however, more than one performer is required. Self-performing devices can be used as an aid to a single performer (e.g., sample looping, sequencing). These types of device will be discussed hereafter. Speaking strictly of a gesture-based performance, the number of simultaneous gesture channels is limited by factors such as the number of limbs a person can have, and their capacity to coordinate all of them while performing independent voices. Although it is possible to use technology to capture as many gestures as there are individual muscles, a human cannot coordinate many different gestures without memorizing the musical performance at the muscular level. One early example of this are one-man band performances, where the performer needs to learn the musical routines to the muscular level. The limits of divergency in solo performances using gestural mapping tools are, hence, related to the human motor coordination limits.

One exploration branch which combines gestural mapping tools with code, can lead very appealing results. Musicians could produce custom programs which handle all the details of composition and performance, and somehow couple their body movements in meaningful ways to the generated musical piece. In these cases, the necessary equilibrium is noticeably delicate between the sense of control, the perception of control, the improvisational freedom and completeness of a musical piece. A performance which is very complete musically, and presents many variation may convey the feeling that the musicians are really following with their gestures what the program requires them to do, instead of them controlling the flow of music, as if they were making the mimic of playing a music which is already playing. This is because a highly complex piece with different modalities needs to recur to timed events, or a highly rehearsed choreography. It would also need to produce more musical events than body events due to the human coordination limitations. If the song presents this high complexity, but the tools is designed not to require a trained choreography, it may convey the feeling that the performer is a mere producer of a random seed to a complex algorithm, because there is no obvious relation between movement and sonic effect. On the other hand, tools that map gestures into sounds in a very direct way, as to make this relation obvious (e.g. air drums) tend to become similar a traditional instrument, not leading into the production of a rich piece, thus needing the integration of more instruments or performers. An excellent example, however, that may have attained this delicate equilibrium is Imogen Heap's performance with embodied controllers. In her demonstration for Wire (Cornish 2013) she

demonstrates the relations between gestures and musical operations. In her demonstration there are examples of live looping, gestural performance of instruments, and live tweaking of effect parameters (Heap 2013). From this demonstration, it appears that her generative system can produce a wide variety of music. The performance, however, still needs to be aligned within an intended track, in a similar way to rehearsed instrumental music. Nevertheless, Imogen Heap demonstrated an interesting approach to produce musical improvisation from gestures and prepared coding, which is an interesting research possibility that needs a long exploration process.

Within conventional music, different levels of divergency are achievable by groups which use gestural mapping paradigm instruments. This has been exemplified by improvisational genres such as Jazz, or even in some western classical compositions framed within the *codas*, the provision of adequate rules for improvisation makes it possible for musicians to improvise while forming part of a group of performers. In these cases, success depends on knowing the other musicians and also the rules about how to perform.

The mentioned improvisational rules sometimes are provided by the style itself, and arguably the structure of electronic music is enough as a rule base for improvisation. For instance, full space for improvisation could be given to a group of electronic instrumentalists, with the conditions that each musician only perform within a musical role (e.g., drums, leads, pads) and that they perform musical brakes in relation to squares of 4 (e.g., a small break every four measures, and a big break every 16 measures). A music improvisation duo called *Skinnerbox* exemplifies electronic music improvisation in groups, resourcing to gestural mapping techniques among other techniques, as it can be seen in the (Hilgenfeld and Gabbai 2017 Skinnerbox Live 2017 video). In the case of performances with more than one participant, divergency is provided by a set of agreed rules, and the capacity for communication among all the participants during the performance.

3.3.2 Sample based performance tools

Sample based performance of music and sound holds an important role in the development of electronic music, most notably in the cases of *Musique concrète* and the appearance of deejaying. These techniques are facilitated by the ability to record and reproduce the recorded material. When it comes to live performance, the two predominant techniques are looping: the repetition of a sound fragment, and playback. Both of these techniques assume additional changes to the sample such as superimposition, re-arrangement and application of sound effects.

The most frequent example of sample based techniques is deejaying; which lends itself for a wide range of divergency levels. It consist of the playback of complete musical pieces or patterns, and intertwining of these pieces by superimposition. The sounds of the tracks can be altered by using signal processing effects, or by manually rotating the vinyl, changing the course of playback. In this way, recordings are treated as tracks. Live superimposition of tracks may be done in more than one way, the most obvious being a sound mix of both. Other examples is the subtraction and mix of different frequency ranges of each piece, or gating, where the volume of each the two superimposed tracks is

switched repeatedly and abruptly in a musical way. This can lead to very divergent performances, where the tracks are completely denaturalized by re-contextualization. The techniques can also lead to very linear performances where the tracks are presented as they are originally to please an audience that reads tracks as social memes.

There are many examples of performances which combine acoustic and gestural instruments with looping, as a way to produce polyphony without additional musicians.[6] For these loop based performances, tools such as Korg Kaoss Pad, Electro-harmonix 45 000's, Boss Loopstations or Ableton are the most recurred. A loop based performance consists on capturing sound patterns that have been produced in the live stage, and reproduce these sounds in constant repetition or *loop*. When a sound is *looping*, the performer can proceed to record other sounds, which will also be captured and looped. This method of performing can convey a great sense of live, since the audience can spectate the gestural performance and henceforth understand the sources of all the sounds which they are hearing.

The attractive aspect of sample-based live performances is related its limitations: the musical score of a sampled sound cannot be changed such way that remains sounding natural. For a fact, the bleeding edge of sample-based modification is recurring to neural networks as a way to trace back sounds to their generation algorithms, with the purpose of re-synthesizing these sounds. Examples of this is the WaveNet (van den Oord, Dieleman, and Zen 2016) and the Nsynth ("Nsynth Super" 2018). But in a composition sense, these experiments are not really sampling tools, but synthesizers that need to be played from notes, like regular synthesizers. Musicians have preferred to harness the unnatural character of re-compositing with samples to produce dramatic effects. Resampling for instance, is what determined, according to (Sullivan 2013, 1–3) the birth of *Dub* Music. Among other examples, this can be heard from nearly all the tracks of The Prodigy's *Experience* album (Howlett, Abram, and Nakajima 1992) in their transposed piano chords, and pitch-shifted voices. This has been the case in many other styles. Another example is *jungle music* whose aesthetic was determined by the nature of old funk drum solos in vinyl records (Butler 2006, 78), and distorted character of reggae lyrics. Apart from the mentioned reference to this, the phenomenon can be clearly listened in tracks such as *Super Sharp Shooter* (Pettit, Ford, and Redpath 2000) or *Original Nuttah* (Wahab Lafta and Williams 2010) among many others. Sample based performance and composition remains a technique with limits that are also their advantage. Certainly sample based music can only offer a gamut of variation techniques that is delimited by the technical capacities of the loopers and their sound-altering operations.

3.3.3 DAW-control based tools

Digital Audio Workstations –abbreviated as DAW– in most cases, share some design attributes and a workflow. Over this paradigm, each different workstation offers some additional improvements to the workflow, while keeping the ability to compose music

6 Two examples of such performances are Beardyman (Foreman 2011) and Reggie Watts (Watts 2013).

through a DAW interface. The DAW workflow consists on having different channels, each of which can contain effects, instruments or sound chunks, often named *samples*. Each channel also possess a lane in a shared timeline, where the *samples* or MIDI events can be placed, as a way to compose the piece. The most used interface for the programming of MIDI events is the piano roll (Fig. 4). An alternative, but less intuitive interface is the one of the trackers (Fig. 5), where the raw MIDI data is presented in a list. Because of its wide adoption, the DAW design paradigm is also used in the design of live performance tools.

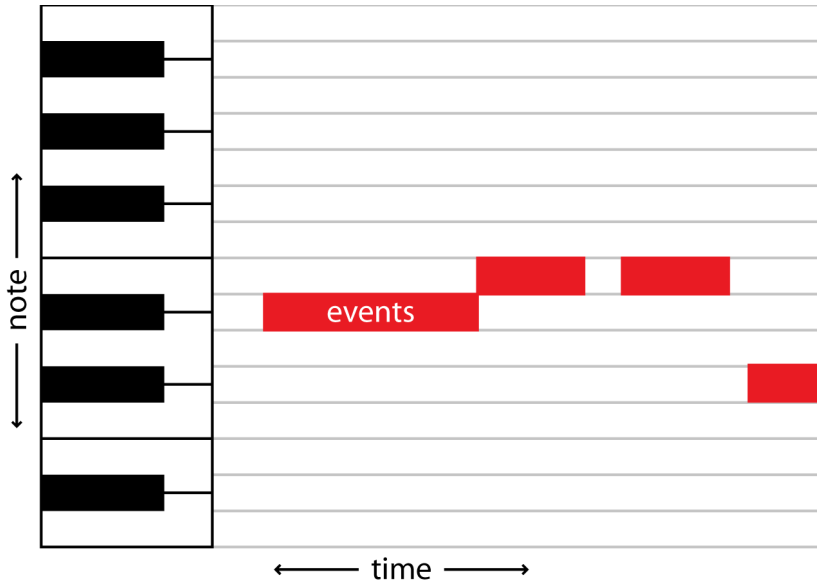


Figure 4: example of a piano roll interface

T.	Event			T.	Event		
00	0x80	0x60	0x70	00	0x80	0x60	0x70
01	0x00	0x00	0x00	01	0x00	0x00	0x00
02	0x00	0x00	0x00	02	0x00	0x00	0x00
03	0x00	0x00	0x00	03	0x00	0x00	0x00
04	0x00	0x00	0x00	04	0x00	0x00	0x00
05	0x00	0x00	0x00	05	0x00	0x00	0x00
06	0x00	0x00	0x00	06	0x00	0x00	0x00
07	0x00	0x00	0x00	07	0x00	0x00	0x00
08	0x00	0x00	0x00	08	0x00	0x00	0x00
09	0x00	0x00	0x00	09	0x00	0x00	0x00
10	0x00	0x00	0x00	10	0x00	0x00	0x00
11	0x00	0x00	0x00	11	0x00	0x00	0x00
12	0x00	0x00	0x00	12	0x00	0x00	0x00

Figure 5: example of a tracker interface

Some DAW composition tools are a DAW in a literal sense. For instance Native Instruments and Ableton have developed controllers that are designed specifically to interface with their own computer-based DAW. This approach takes advantage of the vast computational resources contained in computers, and combines this with a hardware interface that makes musical interaction more fluid and live performance more engaging.

The author of this thesis has been five years performing with *Maschine*. One of the strong points of *Maschine* is the hardware quality, which allows a very fast and expressive input of musical performance. Maschine can be used as a musical instrument with looper capabilities. The interface allows a fast and expressive interaction: there are 16 velocity sensitive pads, 8 encoders for instrument parameters, and certain functions for tweaking patterns. The software design in Maschine, however, bounds the gamut of possible transformations of the loop to a narrow scope: once a loop is recorded in a track, it is very hard or impossible to access individual events and modify their properties. To alter a loop, the easiest is to record it again from scratch, or to access it by using the laptop's interface. The tools for selection and modification of events are very incipient and they do note allow a development of a sequenced loop into a similar one. It is important to note, though, that Maschine has more pattern editing capabilities that most sequencers. Given that Maschine is a controller for a computer host application, however, more transformation capabilities would be expected.

Ableton has been the de facto tool for most of the conventional electronic music performers, regardless of how much of their performance is prepared or played live. In the area of live performance, Ableton's core feature is to have many sound loopers which are tied together in timing. In Ableton's language, these loopers are called *clips*. These clips allow to do an on-the-fly sound or MIDI recording, which will start playing as soon as the record is stopped ("Ableton Manual: Using Push" 2018). Probably one of the most important factors for its success is the fact that the length of these clips adjust automatically to match the recording time, but with a length quantization that is associated to the musical metric. In most of the other tools, the length of the pattern need to be known before recording. The push controller, alike other controllers for Ableton possess a back-lit buttons grid interface. The use of button matrices with as much as 64 buttons is perfect for use as sequencer interface (Arar and Kapur 2013) and an isomorphic keyboard interface among other functions.

Push, being a mere controller of a tool that has been developed for a couple of decades, becomes a vast library of functions for the performance. In tune with the spirit of Ableton, Push intent is to make the most fluent interaction that is possible with the composition. Push, like Maschine allows tweaking parameters of virtual instrument using the hardware's encoders, whose values and labels are represented in a screen with correlated positioning. As Ableton push is much posterior to the development of its host hardware, the mapping of parameters to the interface is much more heterogeneous than most synthesizers and controllers. In Maschine, for instance, each different virtual instrument has well defined parameter to knob association. This results in a more heterogeneous user interface, which affords a broader range of procedures to apply.

It is not easy to track how the idea of using squares to play drums, would become coupled

to the idea of a tactile pixel. One early example of this interface is the *Linn 9000* (Linn 2018) which seems to be the step between a computer-keyboard looking interface and a button matrix interface because it resembles both, a *Linn LM-1* (Linn 2018) and an *Akai Mpc 60* (Linn 2018; and Warner 2017, 160). This can be understood as the link that brought the interface, but this argument stands on an easily refutable position. It can be asserted with confidence, however, that the button pads have become an important, multi dimensional control surface, affording a high bandwidth of input and feedback, (using colours, pressure, position, brightness, texts, among others) that can also present spatial relations (e.g., horizontal time, vertical tone). In the industry, clear examples of a trend started appearing such as Yamaha's Tenori-on or the Korg Kaoss Pad 2. Now a days, it seems that any live improvisation hardware will implement this type of interactive pixel-buttons.

Novation's work with matrix-based composition has been very important to the culture of live conventional electronic music instruments. Circuit synthesizer culturally inherits from Ableton because the interface design of circuit inherits from Novation's Launchpad. With Launchpad, Novation was very successful in the exploration of a user interface that relies entirely on a back-lit button matrix, and Circuit is a later, more evolved realization of that initial interaction concept, now as stand-alone composition interface ("Circuit User Guide" 2017, 5).

If the power of a computer can be scaled down to the size of a cellphone, it makes sense to create a computer-hosted DAW whose host computer is an embedded processor. Novation Circuits are precisely this; a set of digital sequencers with a dedicated digital sound engine and computer. The user interface of the circuit is also matrix composition, thus having a synthesizer with flexible composition interface similar to the one of hosted DAW's. Circuit family comprises *Mono Station* ("Circuit Mono Station" 2018) and *Circuit* ("Circuit" 2018), although it is possible that many new products appear under the same concept.

Some tools simulate the situation of a DAW, with more limited possibilities to facilitate performance. Such is the case with the *groove boxes*. These present a limited set of sounds of different varieties, each one on a MIDI track. The design approach of the groove boxes afford the composition of full pieces integrating drums and synthesizers or samples, and are focused toward live performance of music. Most groove boxes, such as the Electribe, have a sequencer which is very short in features and flexibility. Playing with Electribe consists mostly on tweaking parameters of the synthesizer voices, and by recalling prepared presents from the library. Interestingly, it is this limitation what also brings some insight about good performance tools: dance musicians do not necessarily intend to improvise their performances, and a machine that can recall presets is greatly appreciated in the context of dance music.

The use of groove boxes brings the MIDI protocol into topic. While MIDI has the potential to integrate many synthesizers and sequencers together to build a more complex instrument; by the ways it is implemented in most devices, its function is often relegated to the mere synchronization of clocks. As it can be seen in live setups such as Octave's (*Octave One Boiler Room Moscow Live Set* 2014) among other artists, the groove boxes are used as pre-composed track sources. By using groove boxes or synthesizers, instead of

getting a more complex system, they only get many segregated sound sources which he can only fade in and fade out, but not generate emergent features.

Given the power that is harnessed from using personal computers as host, it comes as a surprise that not everything is possible with personal computer hosted instruments. In a sense, these machines recreate the ideal situation of a studio with unlimited synthesizers, all connected to a single composition system. After analysing all these tools, the flaw of the DAW paradigm seems to be related to the closed nature of design. This idea is most clear with Maschine, where the performer finds a bounded space of what is possible. The bounds of what is possible are defined by the design decisions of the product. This is not a problem on its own, since every system needs a design. The problem is that the DAW paradigm does not offer a framework where original features could emerge naturally from its use. All the *moving parts* –plug-ins, samples, patterns– of a DAW are confined to a space where they cannot transgress its initial workflow.[7] Instruments designed under the DAW paradigm offer bounded options of divergence, and the bounds are defined their design. In other words, they can only do the things for which there is a dedicated procedure.

3.3.4 Modular performance tools

Reactable has been one of the most remarkable *NIME's* in the last years. It consists on virtual environment that works in a modular fashion. Apart from the remarkable user interface, this instrument took advantage of placing the image of the virtual environment's interface in superposition with physical objects, whose positions were tracked back into the environment by using machine-readable codes that they called *fiducials*. This strategy effectively created an environment of augmented reality for modular music composition ("Reactable" 2018).

Reactable propose interesting ideas about the emergence of an environment from a computer system. This case presents a combination of object recognition with basic projection mapping and software which effectively affords modular composition. A similar attempt to do digitally simulated modular composition around the same time, is Block-jam, which considers a signal that travels through the different modules, generating sounds and getting diverted to different paths (Newton-Dunn, H Nakano, and Gibson 2003).

A performance device which is modular-like is Squarp Pyramid. The most remarkable feature of this sequencer, is the non-destructive layers of sequence tweaking that are present such as scale, and the ability to modulate parameters of the events in the same fashion as synthesizer parameter automation ("Squarp Pyramid 64-Track Sequencer" 2016; "Squarp Pyramid Sequencer User Guide" 2016). This allows effectively a more parametric approach to music composition, which implies that many more musical modulations are

7 One example of this in Maschine, is not being able to add a midi effect or applying complex transformations on any sequence.

possible in the domain of the pattern, than with other sequencers.

The idea of a modular synthesis is almost an inherent part of sound circuit design: it would be too challenging to design any functional circuit without discrete components. They developed gradually from research laboratories such as Hermert Eimert and Werner Meyer-Eppler's studio (Warner 2017, 59) where increasingly higher-level sound components were needed, and it is generally understood that later they were brought to massive audiences by Moog company (Warner 2017, 62; Pinch and Trocco 1988). There have been many developments around modular synthesis options such as the Buchla's synthesizers, the ARP 2600 or the E-mu systems. Now a days, perhaps the most varied and developed environment is the Euro-rack, for which new modules and techniques are being developed every day.

Euro-rack environment as a music improvisation platform attains many advantages over the other systems given its openness. Euro-rack in spite of providing a sub-set of the possibilities that circuit design provides, has some unique cultural and technical differences that relates it with very open creative processes. This openness is granted due to its historic independence from a musical or sound canon: Euro-rack standard is born almost by accident with the design of the A100 module. Paraphrasing the author of the system, one of the ideas behind the A-100 system is to allow the use of control signals to control any parameter, without limiting this relation to specified types of signals (Doepfer 2018). According to the *I dream of wires* movie, Doepfer's Euro-rack is related to Don Buchla's concept of modular synthesis, which stood up for free experimentation both, with the design of the modules, and the use of these musically (Fantinatto 2014). The casing and power supply of the standard were inherited from standard casings and power supplies that the designer had in hand at the time, which is the "standard 19" rack system"(Doepfer 2018) standard for electronic cards (Groves 2016). The most distinctive technical values of Euro-rack are voltage controlled parameters, discrete higher-level sound circuits, a connector standard and a circuit board size that enables a standard mounting system.

It is thinkable to consider conventional composition environments such as Reaper or Ableton as modular. It is precisely the difference of Euro-rack modularity against the concept of modularity on these, that make the inherent characteristic of Euro-rack modularity to stand up. For example, in the composition environment of Ableton, there is modularity because many plug-ins can be used in different orders and configurations, in ways were not specifically designed, relying in a host-plug-in scheme. There is also the possibility of plugging different peripherals for user input or output. In contrast, the concept of modularity of Euro-rack consists on a standard of control voltages and an enclosure system that allows any module to take any role, instead of having a framework that leaves spaces where modules will perform a specific role (such as receiving MIDI and outputting sounds). Furthermore, the Euro-rack environment does not provide any predesignated base such as a global clock, or master output. All of these features are meant to be provided –or not– by the modules themselves. For instance while in Ableton a composer is limited to one clock –a necessary limitation for conventional music–, in Euro-rack it is easy to have any amount of different clocks drifting away in their own paces. "These definitions of the various signals, and the distinctions between them –sound

sources and modulation sources– are right in principle, but a modular system like the A-100 often makes a mockery of them. In a modular set-up, all of the modules produce voltages, and can be used as control voltages or triggers, thus blurring the distinction between the various types." (Doepfer 2018) This type of modularity allows for a bigger field of experimentation possibilities. Voltage controlled modular systems, decidedly, present the users with the possibility of making their own synthesizer systems instead of only presenting the possibility of making music directly. Effectively, a *music tool making environment*.

In the domain of computer-hosted modular environment there are also modular systems. Some examples of this are Reaktor and Pure-Data. These two work with lower level abstractions, meaning that the usual operations done by each module are less complex, allowing the user to create a wider variety of systems with it, by using a greater quantity of these modules. User-built modules, however, can be used as modules themselves; which accounts for a usage on a higher abstraction level. These platforms provide the same type of homogeneous modularity as modular synthesizers: signals can be re-purposed, same as some of the modules.

This homogeneous modularity is an ideal example of a platform that allows divergent exploration of music, because instead of having a gamut of possibilities (as offered by DAW-based environments), we now have a field of possibilities. Euro-rack, and analogue-modular music hardware in general allows for experimental music outside the boundaries of our understanding of music, and this has been in general the place for this environment. Clear demonstrations of this, are many modules that foster stochastic composition, such as modules that would capture electromagnetic noise ("Field Kit-Electro Acoustic Workstation" 2018), modules that capture skin capacitance ("New Spikes Milk Edition" 2018), and modules that compose random patterns ("RPG" 2018).

In this sense, some environments that claim being modular, will be considered as non-modular in the scope of this thesis: one example of this are the Roli Blocks ("Blocks: The Instrument That Grows with You" 2018) which despite presenting some physical characteristics of modularity, the composition method is actually based in a DAW or looper model. In this sense, Roli Block modules are mere extensions of one singular access interface to a single composition scheme; in the same way that more than one Maschine hardware can also be connected to control one single running instance of Maschine, or more than one keyboard can be plugged in to one same DAW.

3.3.5 Live-coding performance tools

Programming has been clearly a successful tool to create solutions in many aspects of our lives. Every object that possess some type of programmable data processor reflects that the object's functionality was better represented by computer code. The use of code is so pervasive, that the current use of *analogue* usually serves to refer to what is not digital.

The line between modular composition and live coding composition is not clear when it comes to Virtual-Modular composition tools. Native Instrument's Reaktor, although being

graphically a modular environment, possess some aspects that consider computer processing details which relate the environment to a certain extent, to programming. Additionally, Pure-Data despite its resemblance to a modular environment, is often referred to as a *graphical programming language*. Live programming performances, however, are narrowed down by McLean (2014) to "where source code is edited and interpreted in order to modify and control a running process." (McLean 2014, 63) and most often is associated with text-based coding.

Live programming in concept affords a vast divergency possibilities, even within the area of dance music. One musical trend with this intention, according to Daniel Dylan is the *algorave*, "[i]n essence, the aim is to put programming at the forefront of the club experience, to present the act of live programming as an art form in itself." (Dylan Wray 2013) Algoraver's musical material that is presented as examples in this article features synthesis, sampling and looping techniques.

Live coding for dance music is an interesting proposition, "but to date algorave hasn't[sic] managed to pair the bedroom isolation of coding with the empathy and euphoria of communal club culture" (Dylan Wray 2013) there is still a gap between coded music and clubbing events, which perhaps can be filled with less idealistic programming abstractions to be used in the live context that would allow the creation of those *euphoric* music patterns. Apart from these, live programming is likely to bring interesting new patterns to conventional electronic music genres.

3.3.6 Conclusion

Having produced different discrete categories of tools for musical performance, it was possible to analyse the different ways these could be used to produce divergent live performances. Although each individual product within a category may offer different options for divergency, it was observed that each performance has a defined area of divergency. Coming back to Fig. 2, it is realized that there is not a single axis for divergency. One example of this is the listening of music. In this graphic, listening is not represented as a dot at the left of the spectrum, but as a small range, since, it was assumed that it was possible for listeners to alter their own experience of the musical piece by, for example, focusing the attention on an instrument, or trying to reverse the order of strong and weak beats. This divergency, however, is not achieved in a same way than, say, a musical composition. The production of divergent results in a listening experience is contained within a subjective experience, and the production of divergent results in composition, has an effect in a domain of sonic result. In the case of other activities, such as deejaying, divergent outcomes may not easily be achieved in aspects of composition, but they can be achieved in terms of deejaying. In sampled music, it is not possible to produce musical compositions in detail, but it is possible to improvise the sequence of tracks and their superimposition. In the case of DAW based performances, whose objective is to provide divergency in terms of musical composition, each individual DAW controller offers different modulation options. However, it was found that their divergency (in terms of musical composition outcome) needs to be limited by their design specification. Two performance paradigms, however that did not have this inherent limitation, but only

existed in a practical sense, are the modular and live programming environments: while it is virtually possible to make any musical performance and modulation from these types of system, there are some current practical limitations.

3.4 Thesis statement

Regarding current tools for conventional live electronic music, there seems to be a space for divergency, in terms of composition and transformation, which is limited. For most part, dance electronic musicians are surprisingly attached to performances where all the musical material is prepared beforehand. This provides them the ability to provide any musical modulation (since it was carefully composed beforehand), and a safeguard against performance mistakes. However, this also challenges the *live* and *collective* sense of the music performance. Often the live-ness of the performance is relegated to tweaking of a sound parameter, or a change on how many times the same loop is repeated with respect to a studio version. Collectivity of the performance, is often relegated to the mere fact of sharing the physical and sonic space. It comes as a surprise given all the available technologies, that music making tools are still offering the musicians with the same loop-based paradigms, with limited modulation algorithms. In most cases, the improvised parts of musical performances need to be very simple patterns within the constrained possibilities of a software. This thesis intends to contribute with the production of one new mean of satisfactorily improvising *conventional live electronic music* without needing prepared musical material, and allowing exploration of composition aspects at the performance time.

The topics explained to this point seem to avail the idea of a tool with more potential for divergency. Theoretically, such tool would be well appreciated under the reading of some electronic music related value systems such as the *underground* electronic music genre and close the circle of live-performance versus collectiveness in certain contexts of electronic music performance. The key aspect to success to this project, therefore will be defined by divergency, affordance of composition, and affordance to modulate this composition. Specifically on how the product affords *fluency*, *originality* and *flexibility* in the live performance of *conventional electronic music*.

4 Development & production

This chapter explains the development process of what will become the Virtual-Modular environment. It starts with an outline of the design process, where a reflection about the theoretical frame lead to the fundamental concepts and processes that will be used in the production process. In a sense, the definition of the design concept works as a theoretical frame, but this time, it is established with a specific conceptual solution in mind. The explorative design process starts from the *fundamental level*, meaning that it creates the most basic rules of the intended *music environment*. During the development of this project, two design processes took place in parallel: the design of the Calculeitor controller, which is a hardware and the development of the *modular environment*, which is an idea that gets tested by the creation of the *Virtual-Modular environment*. Relative to these items, the idea of the *modular environment* is established in the *Fundamental level explorations* chapter; the development of the hardware is explained in the *development of calculeitor* chapter, and finally, the development of the *Virtual-Modular environment* gets explained at the *exploratory iteration in the Virtual-Modular environment* section. Finally, this chapter contains a section where the potential of the design concept is explored, as if its development was sustained more years in the future.

4.1 Outline of the design process

Given that there are no known specifications for the end product, the development of this thesis was based on iteration. Iterative processes are very common in product development. They are inherently divergent, consisting of a repetitive application of gradual changes to a solution. Each iteration consists on design, testing and evaluation or analysis (Laurel 2003, 176) as displayed in Fig. 6. Following an iterative process is a double edged sword because the scope of possibilities to be explored is limited. The negative aspect of this is that the method does not guarantee an arrival to the best possible solution since not all the possible solutions can be by the heuristic. The positive aspect is that it allows the creation of solutions where other heuristics could take potentially infinite time.

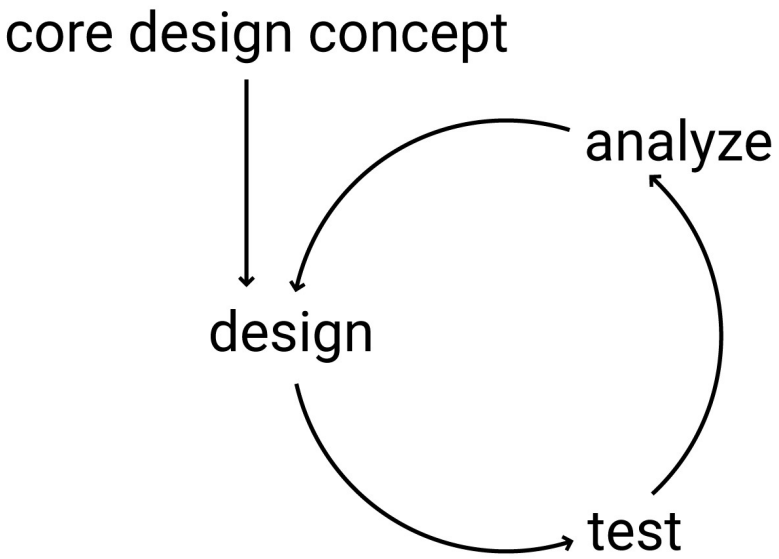


Figure 6: Iterative design process Laurel (2003), p.176

4.2 Definition of the design concept

Examples of composition environments such as Pure-Data, Euro-rack, or the mere idea of modular composition are based on a concept which is interesting to this project as mean to achieve divergence in music composition. These environments have served as tools for sound and music experimenters to create sounds that were previously not possible using

traditional instruments. The initial enthusiasm to explore these sounds, was because they were generated electronically which was a novelty. The fact, however, that this exploration is still occurring is because the modular environments allow the users to construct their own synthesizers and composition systems. Hence, as a composition or performance paradigm, modularity offers an additional dimension of divergency in comparison to mechanical instruments: the ability to alter the behaviour of the instruments in real time. This modularity is henceforth used in this project as the design core concept.

4.2.1 The three domains: environment, system and music

For a modularity to exist in the same sense as modular synthesizers or programming languages, *environment* is a crucial base concept. One of the first ideas that comes to mind by the mention of the term *environment* is the Earth's *ecosystem*, the environment of living things. Organic living systems can only express in the context of the physical world, (Maturana and Varela 1980, 1994) across a concrete set of dimensions, and given the environment and the nature of the systems, they are characterized by a certain set of rules. A living organic system would not make sense, for example, as a computer program and vice versa. In this case the intention is not to refer to environment in the sense of the musician's presence in the ecosystem, as Waters (2007) would. This accounts for a complete definition of a musical performance as species that live in an ecosystem. For the design of modular composition systems, however, a broader idea of environment is needed which is not limited to ecosystem. The idea of ecosystem, also conveying the idea of a framework of interrelations between elements, could be considered like a particular manifestation of environment. This opposes to Waters's (2007) subordination of environment to ecosystem (where environment is one of the parts that form the ecosystem). Environment herein will be considered as a conceptual –or perceivable– system which is capable of containing systems, allowing these systems and their parts to interrelate, and provide means to produce and organize these systems. Under this concept, henceforth, ecosystem is one possible instance of an environment. This view of environment is shared within the field of computer programming; with the so-called *programming environments*, which incidentally may the second idea that comes to mind when thinking about environment. Some other examples of environment are Euro-rack and Pure-Data, where a set of rules facilitate the emergence of synthesis systems. Furthermore, it is possible to consider the domain of mechanical construction as an environment which facilitates a certain gamut of musical systems (instruments) among other things. These last three examples are mentioned at the centre of the Fig. 7.

Environments therefore, are defined here as the means of production and manifestation of a system. In the case of music, environments do not directly create music[8] but instead afford the creation of music making tools, as for example a particular synthesizer in the environment of electronics. This thesis will therefore refer to environment as a system

8 As an illustration, many programming languages cannot produce music directly. Instead it is possible to build music making systems with them.

that is intended for the creation and use of other systems. Under this definition, examples of environment include programming languages, modular synthesizers and building toys.

The design of a composition environment implies three different design outcomes or design layers, as represented in Fig. 7. The last and least abstract layer is the musical outcome. The gamut possible musical outcomes is delimited by the affordance of the music creation system. This relates the musical outcome layer to the second layer: the design of music making systems. The range of possible musical systems, again, is limited by a lower layer. This brings us to the first layer, which is the environment design itself. Defining this three-layer design challenge reveals the radical difference between an environment design and a music tool design. While the design of a music-making tool affords a number of musical outcomes, the design of an environment, allows a number of possible musical outcomes times a number of possible musical systems.

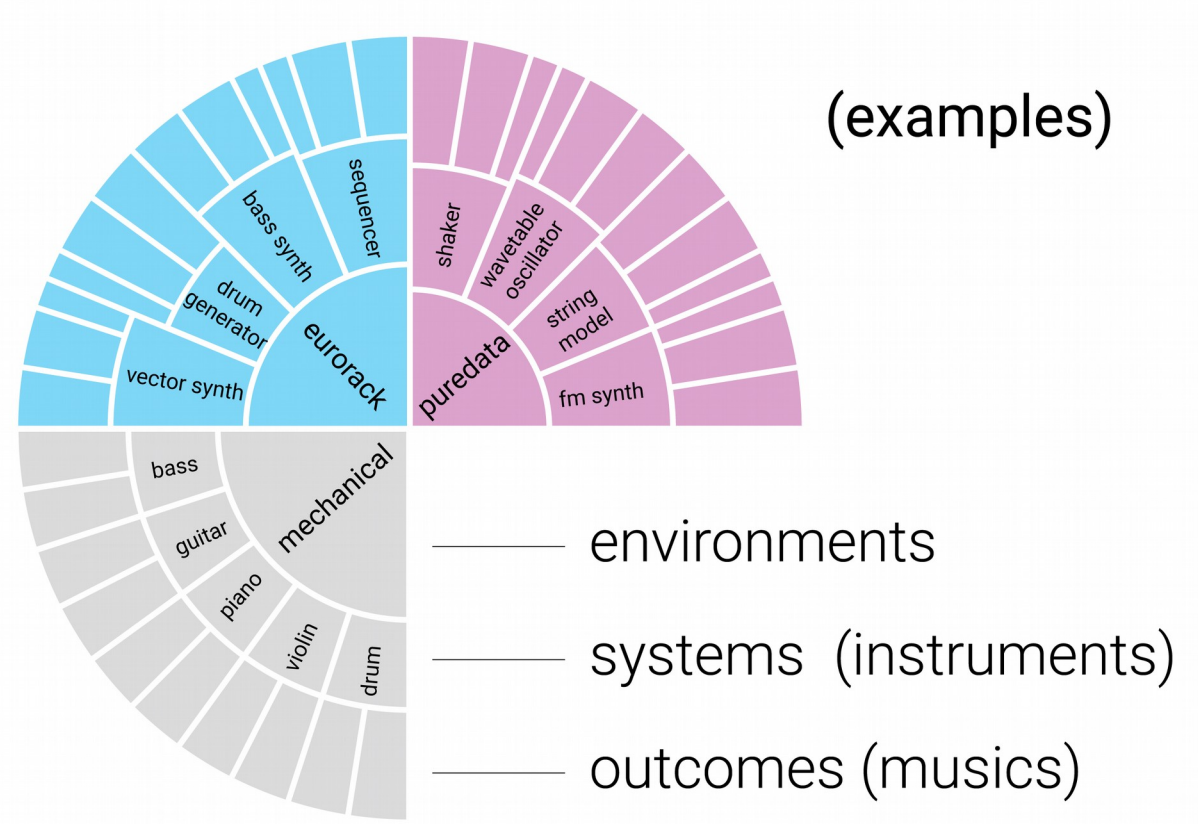


Figure 7: Three design and outcome layers

Each of these layers may be referred to as a different domain, meaning that each layer can have a different set of terms, and each term can have a different meaning depending on which domain is being analysed. For example, deficient user interface feedback in the musical domain does not imply that there is a defective user interface in the system domain. In practical terms, the system that is producing the music may be very clear because the interface is representing it in an intuitive way, but the way that this system is representing the musical outcome may be inadequate. The environment will be developed

upon the affordance of programming languages and digital electronics, which are a perfect fit to describe the intended discrete nature of the environment.

Having defined the concept of a three-layered musical environment, it is now possible to outline the project's exploration process, as shown in Fig. 8. It is possible to start with known live-composition elements, such as sequencers, and use them as modules. The ability of sequencers to form part of a modular network is proven by their use in analogue modular environments. In this case, prototypes of sequencers were programmed in javascript, with the ability to exchange digital signals. This experiment revealed a method of how to break down composition elements into sub-units that can be used as building blocks to many other composition modules. Finally, in order to produce meaningful conventional live electronic music performances, higher-level modules (building blocks) were designed. These made it possible to create improvised music. Each of these stages did proceed as iterative processes, starting from a preconceived idea, and changing in accordance with testing.

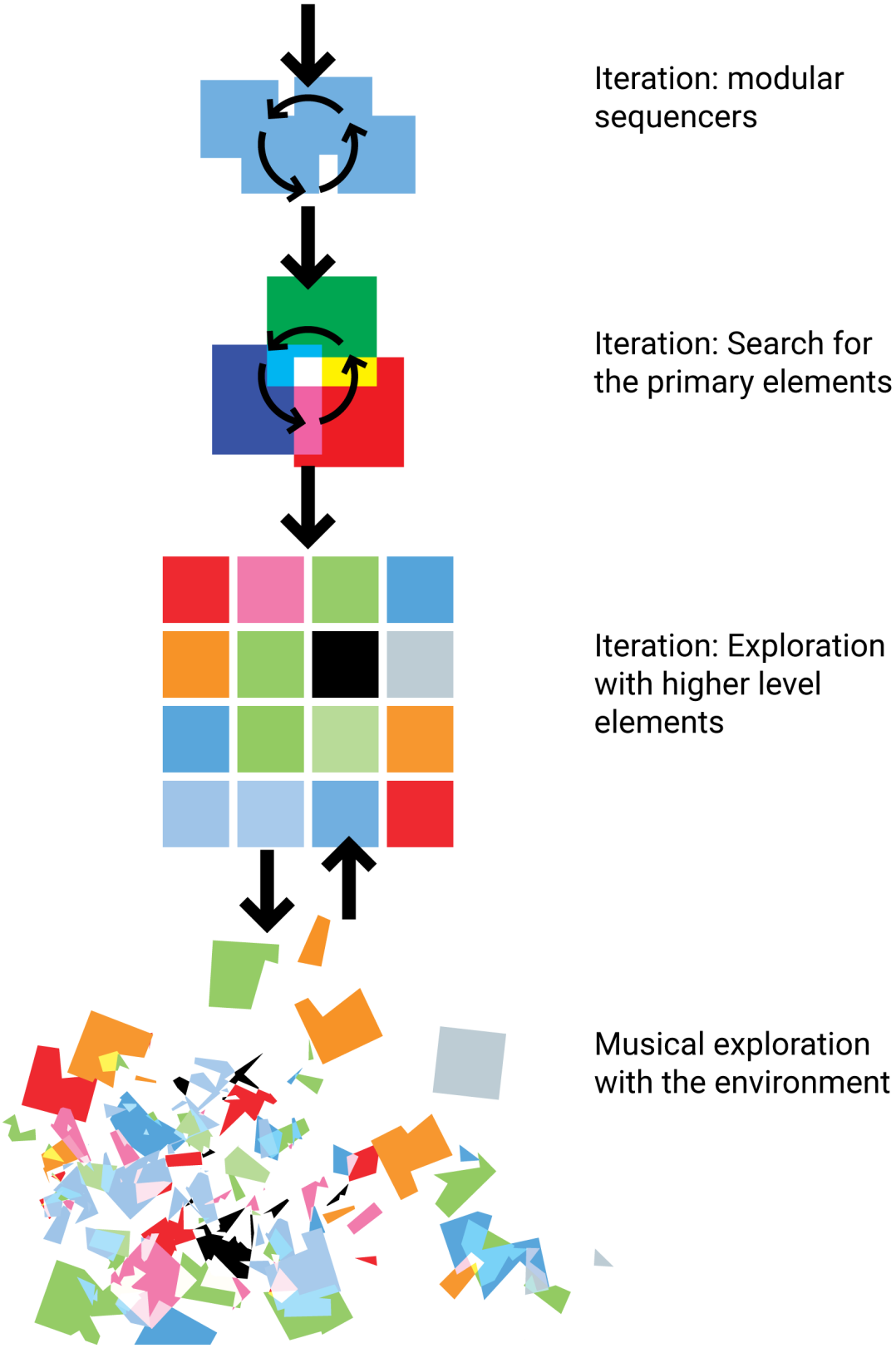


Figure 8: Representation of the environment design process as an analogy

4.2.2 Event-messages as a communication medium

A strong reason for Euro-rack being a good tool for experimental sound performances is its freedom from musical structures. This same reason, however, explains the limitations of the same tool for the composition of conventional electronic music. It is related to the type of abstractions upon which these work. The composer's overwhelming preference for systems which are limited in divergency, such as DAW based solutions (like Ableton or Maschine) is related to their composition abstractions being based on conventional musical concepts such as notes, arrangements and chords. Working with these abstractions makes the composition of conventional electronic music easier. On the other hand more divergent, modular composition environments possess continuous abstractions (e.g., voltage, clicks, transients). With modular synth environments, expertise is required to achieve musically conventional results. One example of this in the Euro-rack environment, is the difficulty to achieve polyphony: while polyphony is a strongly expected feature on conventional music, it can only be achieved by having many copies of each analogue voice, or by using a digital synthesizer system. Although current modular sound compositions systems can achieve musically conventional results, they are not designed with this function in mind, and tend to make this task harder.

Other conventional music representation which is problematic by using control-voltages, is notes. Instead, in Euro-rack, tone (a continuous expression of frequency) is usually represented by a voltage on a scale of one volt per octave (Doepfer 2018). This has the advantage of being able to represent tones outside conventional scales. This representation, however, can become problematic when it comes to more conventional compositions. In this case, physical effects such as thermal coupling or electromagnetic interference could lead compositions to go out of tune.

When it comes to the improvisation of more conventional music, a musician will need discrete events in order to represent abstractions such as notes and scales. In particular, the communication between modules needs to allow the coexistence of more than one information bit at a time, something that Euro-rack modularity does not allow given its continuous, analogue electronic signal paradigm. By contrast, the parametric composition nature offered by Squarp Pyramid inspires the creation of a system that manipulates musical events, with a communication protocol similar to MIDI. This allows the expression of notes, polyphony multiple-voices and other abstractions of a discrete nature. The task ahead consists of specifying a nature of this language outside a strict standard like MIDI.

A concrete example of the frontier between event-composition and signal-composition can be observed in the Pure-Data environment. Pure-Data contains two different types of signal that can propagate: one type consists of values, symbols and bangs while other type are sound signals, which need treated differently because of the different types of processing that each requires. In the context of Pure-Data:

> *[T]he thin [connections] . . . are for carrying sporadic messages, and the thicker ones (connecting the oscillator, the multiplier, and the output dac~ object) carry digital audio signals. Since Pure-Data is a real-time program, the audio signals flow in a*

> *continuous stream. On the other hand, the sporadic messages appear at specific but possibly unpredictable instants in time.* (Puckette 2006, 17)

A signal is a continuous stream of a continuous value or its simulation. Examples of signals are the voltage level on any cable of a modular synthesizer, a sound buffer, or the position of a knob or fader on a control panel. An oscilloscope view of a possible signal is exemplified in Fig. 9. An ideal signal represents with perfect precision the state of an analogue output and is sustained for as long as the output remains in that state. Real signals, however, are subject to problems such as radio-frequency noises, thermal coupling, hardware defects, resistance in a cable and capacitance. A signal is therefore best suited to represent events of continuity.

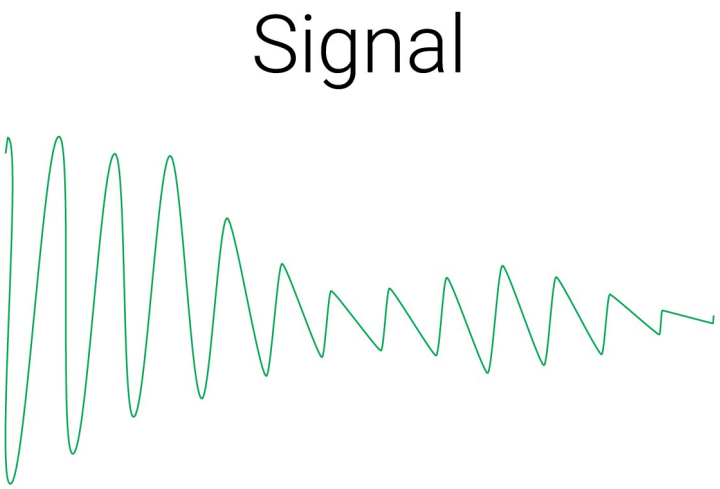


Figure 9: signal example

An event that occurs at a moment in time, containing one or more discrete values hereafter will be called an event-message. Some examples of event-messages include MIDI, UDP packets and telegram messages. They are used most often to control discrete behaviours, such as states, tones, scales and metrics. The manifestation of event-messages is numeric, as exemplified in Fig. 10. An event-message, manifestation thus, will best represent events of discrete behaviour.

The distinction between event-messages and signals seems similar to the distinction between continuous and discrete signals; these new names were introduced to specify this distinction into the domain of modular composition. For example, a Euro-rack clock signal could be thought as a discrete signal, but for this case study such signal needs to be in the category of signals. on the other hand, with different eyes, an event-message signal could be considered as continuous when it is being transmitted via wire. A need to specify the distinction needs to be made, based on the intention of this signal in a musical environment. This distinction, as observed, takes a slightly different meaning than

speaking of continuous and discrete as an engineering or physical aspect.

The new distinction also allows for the definition of an ideal signal and an ideal event-message: where a signal ideally spans along a certain amount of time to represent some value (such as envelope), an ideal event-message would happen in no time. This implies that, whereas a signal defines a timed event in a continuous timeline, an event-message divides time between before and after, ideally taking zero time. To make distinctions between event-messages and Signals, the account is for the intention of the signal rather than its actual continuous or discrete type.

Figure 10: event example

The concept of event-message is useful to this project because, as explained before, continuous signals have served for divergent composition of experimental sound, whereas discrete event-messages can express conventional music abstractions. These conventional music abstractions in the context of a modular composition system ultimately are expected to allow a modular system to produce conventional music, as it will be explored in the following chapters.

4.3 Fundamental level explorations

There were two preceding explorations done by the author which helped set the interest for the present thesis. One of them, proposed the idea of creating a musical building tool as an analogy to building blocks, called Brocs (Aldunate Infante 2013b, 2013a). This exploration opened the interest in musical composition as construction of systems, which led to a second, virtual implementation of a similar nature named Licog composer. The first, being a thesis project, led to a concrete product of physical nature. The second project, being an exploration without a purpose, has less defined boundaries and was

implemented twice using *Processing* language, and later one incomplete attempt was started using javascript (Aldunate Infante 2014).

During these explorations, two naming conventions came naturally to the dialogue. Given that each component has inputs and outputs, modular systems of this kind have an inherent direction. By the words *down* and *up*, if referring to the order of signal propagation. A module *upper* with respect to one other module is meaning that the output of that module is connected to the input of the refereed module. The same in the opposite case: a module which is *down* the patch, is receiving signals from the refereed module. The same idea can be explained with the analogy of *parent* and *child* sequencers, where the analogy still refers to the hierarchy of connections in cases the *up* and *down*.

The mentioned building blocks represent the most basic components in the ambit of conventional music, namely notes and discrete events. In the context of this thesis, the environment paradigm where each component is a single sound event will be referred to as *molecular*. By developing these molecular environments, some interesting emergent features were discovered, which ultimately motivated an ongoing exploration. Nevertheless, a hardware version of such device is still commercially challenging, because of the high costs of having many copies of a micro-controller based component, and the difficulties that pose interconnecting the necessarily large quantity of these together. This, together with the vast area that was left unexplored in the previous experiences, are the reasons why the *molecular* paradigm remains as a reference rather than a complete specification for the development of this thesis project.

4.3.1 Composite elements environments

The first exploration on how to define a modular composition environment consisted in the design a module with the behaviour of a sequencer which could be instanced many times in a simulated environment. The sequencers in this virtual environment had connection nodes that allowed them to communicate. This module was largely based on real life analogue sequencers, an important higher-level composition device in most modular systems (e.g., Euro-rack, Moog Modular.). The difference is that these were simulated by using a programming language, meaning that all the sequencer functions and effects were digital.

In this exploration the interest was on the amount of different systems that could be built with a small variety and amount of modules. Another topic of interest was the manoeuvrability of these systems from a composition point of view. It was not important whether the system would comply with these metrics to the full extent, but rather whether it would display a potential on those aspects. In other words, the idea was to explore with very basic modules in order to imagine future development directions. In addition to the creative exploration it was interesting to note different design challenges that emerge naturally from the idea of a modular digital system.

From that starting point, many factors of their design were subject to changes as they are

adapted from an analogue to a digital environment. One design challenge was defining which parameters need to be user-defined, or what type of messages would them be exchanging, and how they would react to these messages. The intention is to explore the possibilities of adding modular behaviours to a sequencer and understanding how a module that can generate music on its own, can also attain emergent features when they are taking part in a network.

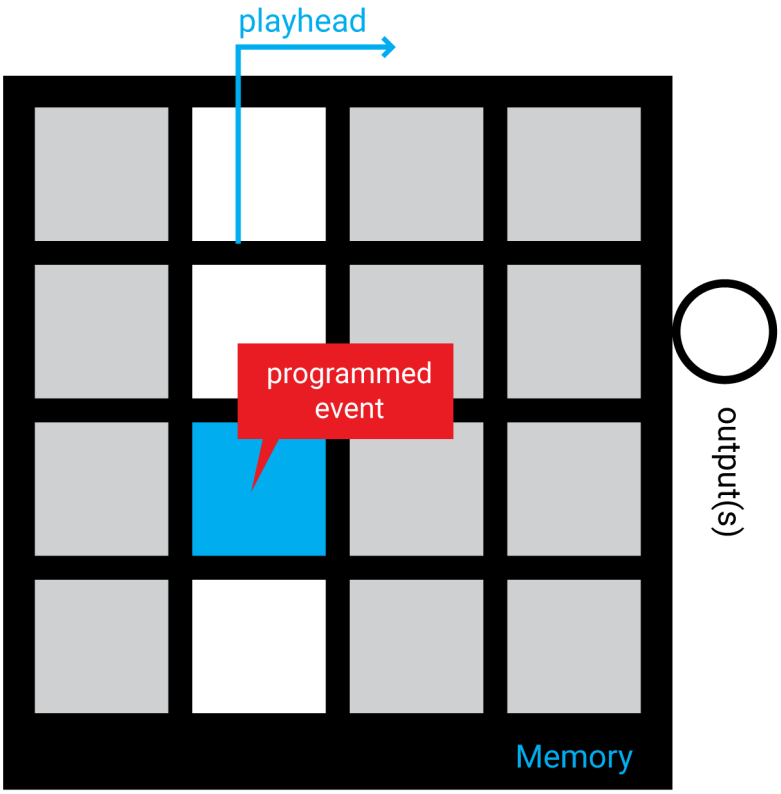


Figure 11: representation of a mono-sequencer

A basic simulated environment for modular elements was programmed using javascript. It defines a graphical user interface, and a module that can be instanced multiple times, which gets graphically represented in the mentioned interface. It contains a layer on top that defines behaviours such as interaction, response to messages and user defined behaviour options. The Fig. 12 is a snapshot of this javascript prototype.

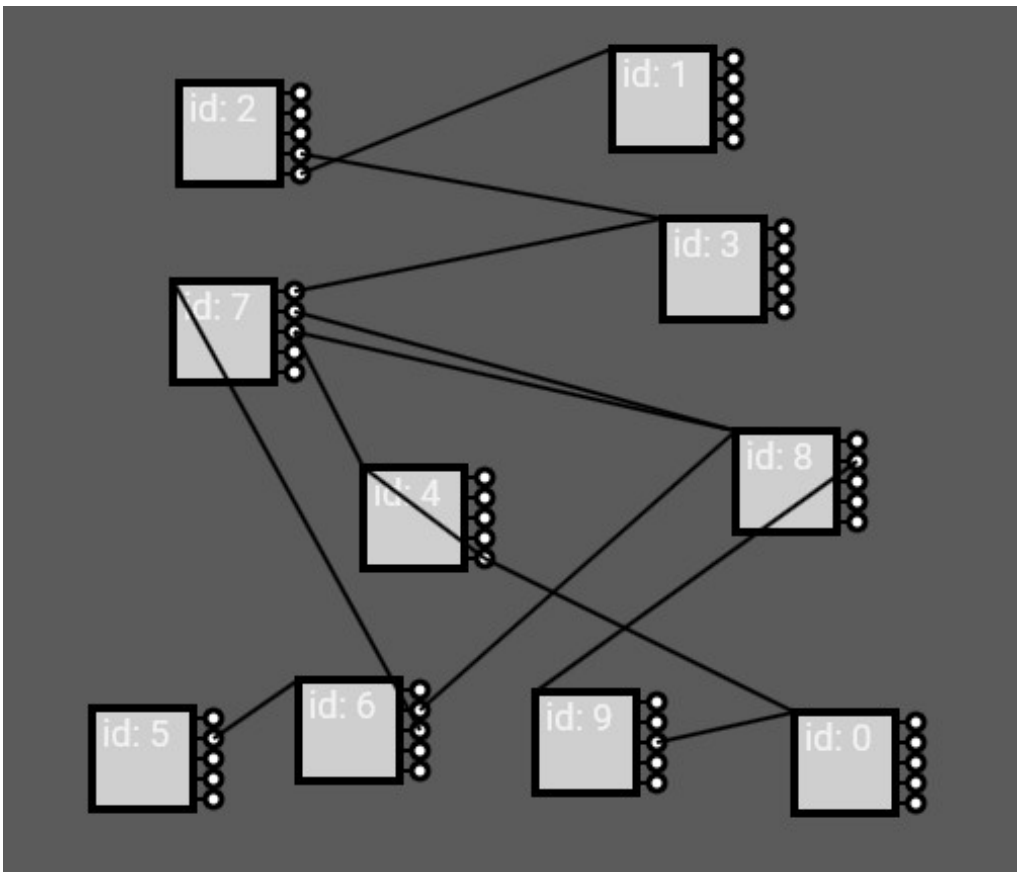


Figure 12: snapshot of the experimentation modular environment

The first exploration led to the idea of using a sequencer as an event-to-event mapping matrix. The first prototype of a *mono-sequencer* treated the horizontal axis as a time axis, and the vertical axis as different *voices*, making it a 4 voice, 4 steps sequencer. First, these responded sequentially to a global clock, and in a second attempt, their *play head* would only change in response to signals that would come programmed from a parent sequencer. A *clock active* setting needed to be added, however: if none of the sequencers is being triggered by a clock or a user input, there would be no original event to propagate in the first place.

This configuration permits a sequencer to be re-purposed as an event re-mapper: if a sequencer sends a `[0,1,2,3]` sequence, the child sequencer would play as a normal sequencer, but any other sequence such as `[3,1,2,1]` will cause the child sequencer to play in non-sequential order (as illustrated in Fig. 13). In this way, the lower sequencer matrix becomes a matrix that maps input signals to output signals.

A usage example of this feature would be to create a palette of notes in a scale that are sequenced by the parent sequencer. Or perhaps, a palette of chords. It already presents us with an improvement over the traditional sequencing approach because, if a musician wanted to change the harmony of a melody, instead of needing to reprogram every note on each step, it would be possible to re-map the musical scale by changing one event per tone. This approach also allows complete transformations to a melody, if for example the user starts mapping all the child sequencer events to a same note, while the parent

sequencer is playing a sequence with many distinct notes, and then start adding tonal variations, thus obtaining a melodic progression which was not possible before in most digital sequencers.

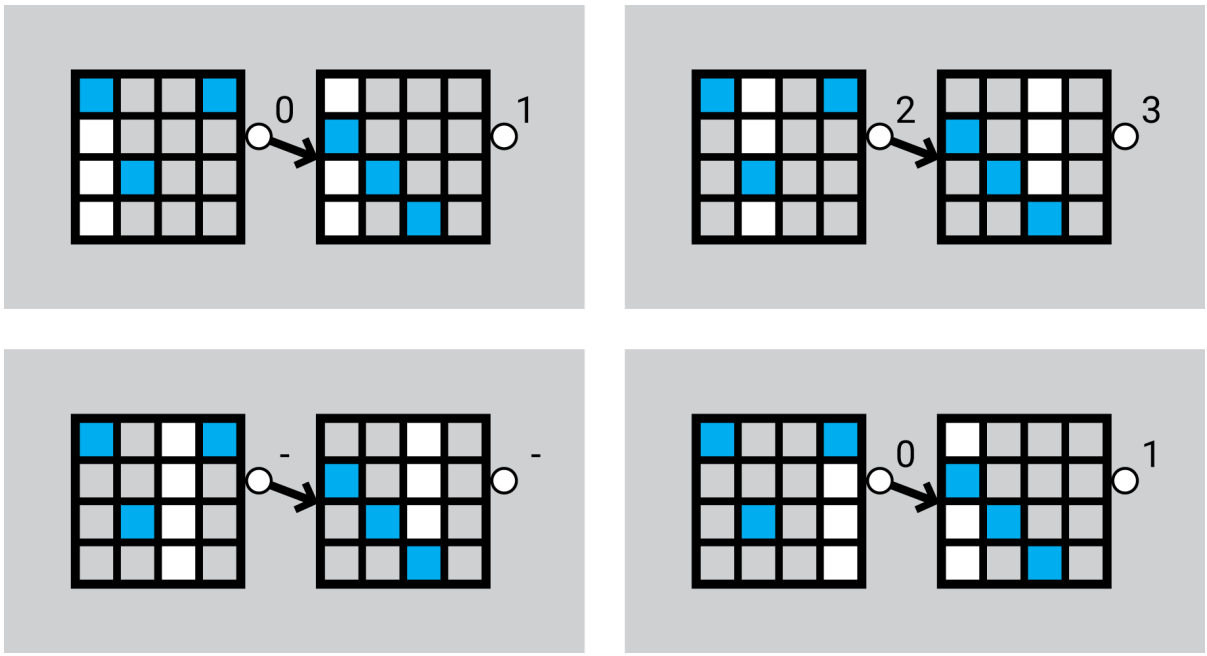


Figure 13: example: the sequence 0,1,2,3 is being remapped to 0,2,-,0, and then to 1,3,-,1

Each possible sequencer value (vertical axis) of these sequencers corresponded to a different output node. This permitted the route of an event to change from one path to another depending on the step: an effect similar to what can be done by using more than one analogue sequencers, if they have dedicated step outputs (such as the Korg sq-10).

Each of these sequencers has an *id* number that corresponds to the order at which sequencer was created. An interesting emerging problem is that some behaviours may be different depending whether the connection goes against the order of the id numbers or with the order of the id numbers. To exemplify: if sequencer n°2 is parent of sequencer n°1 is against the id order, and the inverse order of connection would be *with* the id order. This is because the id also dictates the order at which each sequencer's internal functions are processed by the computer's processor. If each module is set to respond instantly to any signal, there is no big difference on the response regardless of whether the connection goes *up* or *down* the id. Only makes difference with respect to what output number each child is connected to similar to Pure-Data (Puckette 2006, 212). But if the modules are set to wait for a clock step to respond, there will be a difference: if a connection goes *up* the id number, upon clock tick, the module will have already received the signal to which it has to respond at clock time. If the connection goes *down* an id when the clock ticks, however, the parent would have not yet sent the signal to which the sequencer has to respond, and therefore, it will not respond until the next clock tick, adding a delay. This problem resembles the one of digital systems design, and is the reason why a processor that has millions of transistors, cannot make more than one sequential operation per clock tick

(Vahid 2007), which is contrasted by how a sound signal can go through a full analogue process at virtually the same speed electricity travels across the wire, as seen in *field programmable gate arrays*.

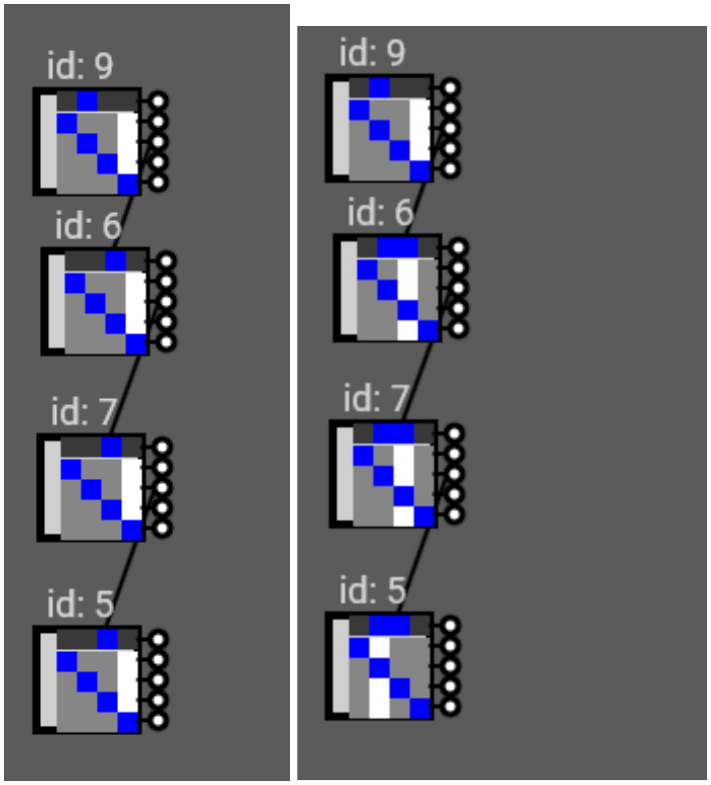


Figure 14: a. a — instant response generates a negligible time difference regarding response up and down id's., b — When elements are clock bound, down-id connected elements will be one clock behind.

This is an interesting problem for which a solution is needed: if this was a hardware situation, there would be no clear rule, because the elements would not be updated progressively as in the computer simulation. The result is that instead of a clear timing rule, whether the response is delayed or not will depend on the tiny difference of time each processor takes to receive and respond to a signal.

The first proposition that was tried, consists in that a module, although receives and reacts instantly to all incoming signals, it buffers all the resulting signals into an output buffer, is set to be sent in the next clock tick. The second attempted solution consisted on processing all the elements in two separate processes, in the same a software would treat the drawing of graphic layers if it wanted to ensure that elements to be drawn from an array, would be drawn in a different order than the one specified by the array. The problem that emerges from applying the first solution, is that the delay still happens, but in an even less intuitive way: the delayed reaction that is caused by sending a signal to a module with lower id number is relegated to that child module, making the cause of the phenomenon less understandable. A similar behaviour results when trying using the same type of buffer for the inputs instead of the outputs. The second solution idea was applied by giving to each element two signal queues: one queue for the incoming messages, and

other queue for the outgoing messages. Upon clock, all outgoing messages are sent, and after clock, all incoming messages are processed, thus generating a new set of outgoing messages, effectively generating a layering of time. This approach generated a consistent behaviour of delaying the signal propagation 1 clock per connection, as seen in Fig. 15.

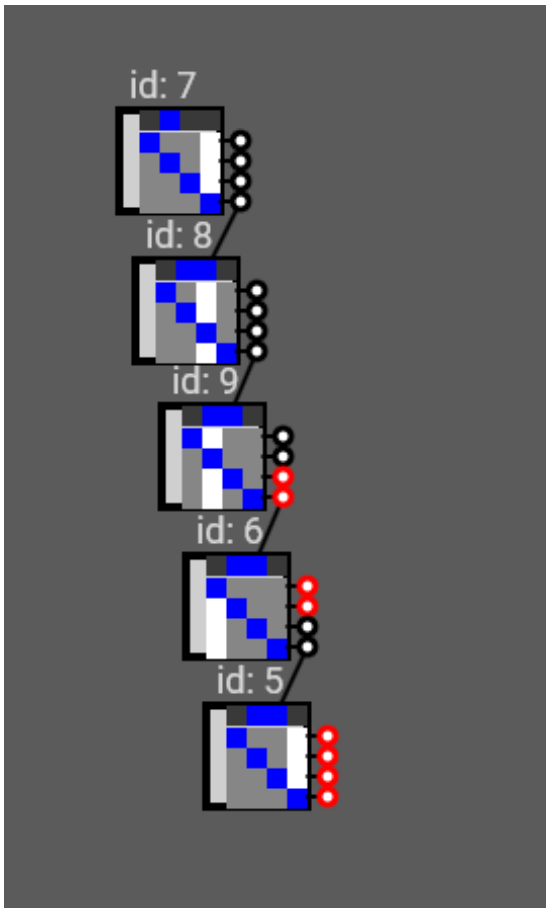


Figure 15: Demonstration of a consistently delayed signal by one clock per connection

The second solution mentioned, however, comprises adding a whole clock delay for each node. This compromise reflects that the system needs not to be intended as globally clock synced except for some modules that are clock-based, such as a sequencer. In a modular system whose modules may need to be coordinated to a clock signal, to be two distinct types of processes need to coexist: the processes which accumulate tasks until the next clock tick, and the processes which respond instantly, regardless of the clock. In this way, it must be expected that signals flowing from one clocked device to another, will obtain a delay in a way analogous to *micro-controllers*. Signals going through a non-clocked path, on the other hand, get processed as soon as possible, in a way which is analogous to a *field programmable gate array*.

There are some other clear interesting features that suggest lines for further development. For instance, by extending the capability of each of these mono-sequencers to a complete sequencer, many other expressive manipulations would be possible than the ones offered by isolated sequencers. One example is that the signal emitted from one sequencer to

another could be comprised of many numbers (in this exploration the communications were limited to single numbers) in such a way that a static message could be transmitted and routed through many sequencers. In these polyphonic devices, some bytes could be intended as destination messages which dictate how to route and transform the message, while some payload bytes may go through the whole patch sometimes altered and sometimes forwarded until a destination (e.g., synthesizer). This will provide with a concept of multi-layered message processing: one layer which determines the physical route taken by a message, and other layers that determine the effect of this message once it arrives to the final destination. In this way one could use these as modules as if they expanded a single sequencing interface[9], and still be able work with them as modules that expand the capability of the system, as in modular environments.

4.3.2 Finding the primary elements of the environment

For the design of an environment, it seems impossible to define the perfect specifications because it is unknown what the future elements, or building blocks will require from the environment to be possible. Poor definitions of an environment could lead to excessive compromises in versatility, and may disallow the existence of certain components. For this, a particular iterative method was devised. This method allows to discover the desired basic building blocks for any given environment that aims to afford the creation of a certain set of systems, and by consequence, define some generic characteristics of the environment which host these building blocks. The use of this process led to a good set of specifications that proved useful for the environment being sought.

Given an initial set of systems that the environment is supposed to enable, the method allows to break down these system into increasingly basic sub-units until left with a minimal set of different units. It is expected that the resulting parts can be used build any of the initial systems on the set. An example: if the objective is to make a system of parts and pieces that could be used to build any transportation machine, an initial set of systems would be a set of transportation machines. To make this process iterative, the initial set of systems are not considered any more like systems, but like *units*, which can be potentially made of other sub-units.

The first step is to conceptually explode the current components, into sub-components that permit building easily any of the initial components (e.g., motors, wheels). This is what appears in Fig. 16 as the *divide* transition between *components* and *sub components*. If this was the only step to be iterated, it would lead to a set of sub components that can effectively build any of the initial set of components, but probably many those components will be compatible with one and only one of the initial components. This is why the second step consists on finding similarities among those sub-components: one sub component may be adapted to comply the same function as two-sub components, thus reducing the amount of components and making each sub-component more general-purpose. This also leads to a standardization in the way the components connect one to

9 alike Roli Blocks (“Blocks: The Instrument That Grows with You” 2018).

another, which leads to a third necessary step: homogenizing the ways to bind or connect those components together. The third step could be thought as part of the second step in the sense that communication routines can also be considered sub-components. This is seen in Fig. 16, in the arrow that points down from the *sub components*. Each iteration consists on taking the sub-components as the new components, and repeating the process, as expressed in the remaining arrow of Fig. 16. Doing this process for enough iterations lead to a certain set of general-purpose components, and hopefully very few components that are specific. The interesting part is that using the general-purpose components that result from the operation, new components can be built that extend the possibilities of the initial set (Figs. 17, 18).

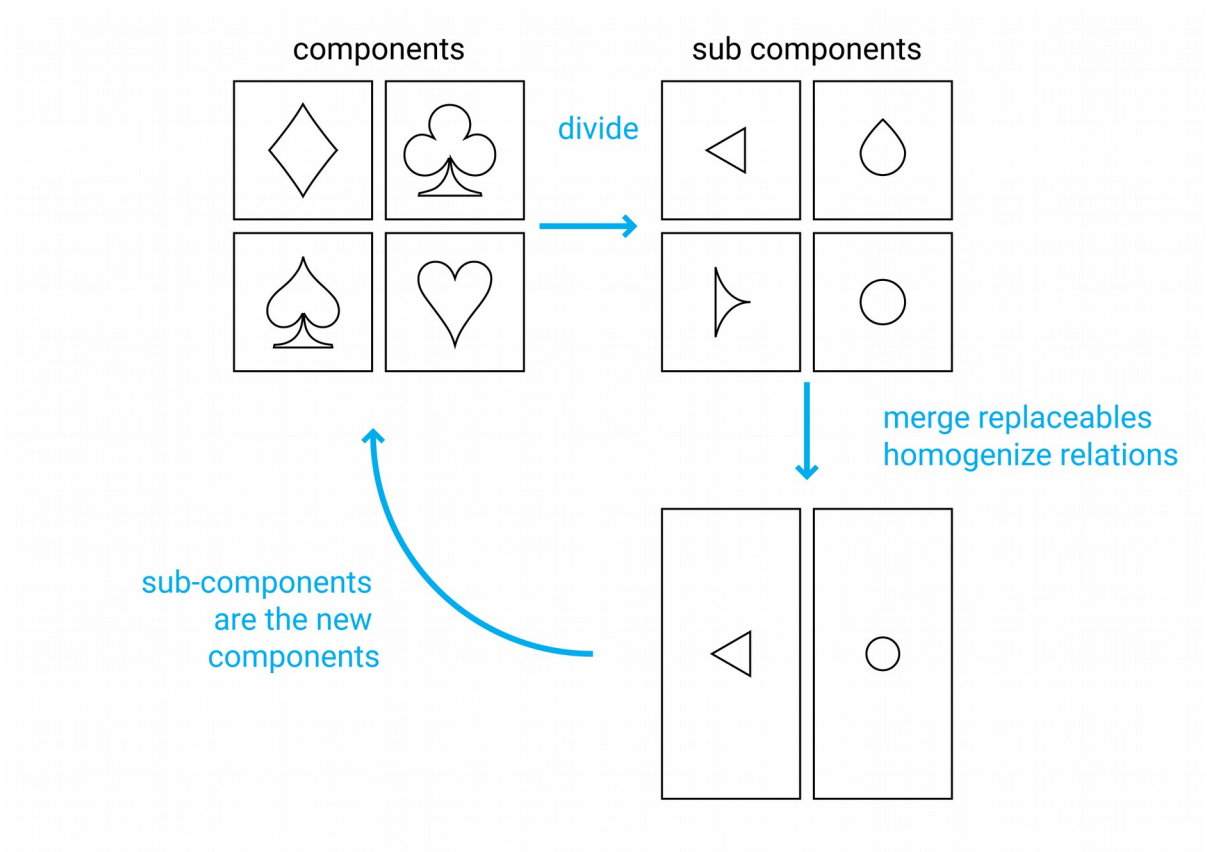


Figure 16: graph of the iterative process

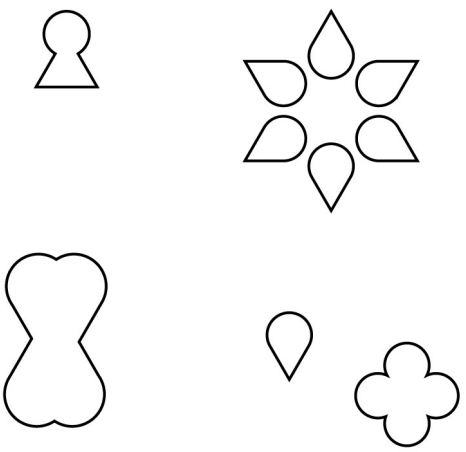


Figure 17: Example of emerging components

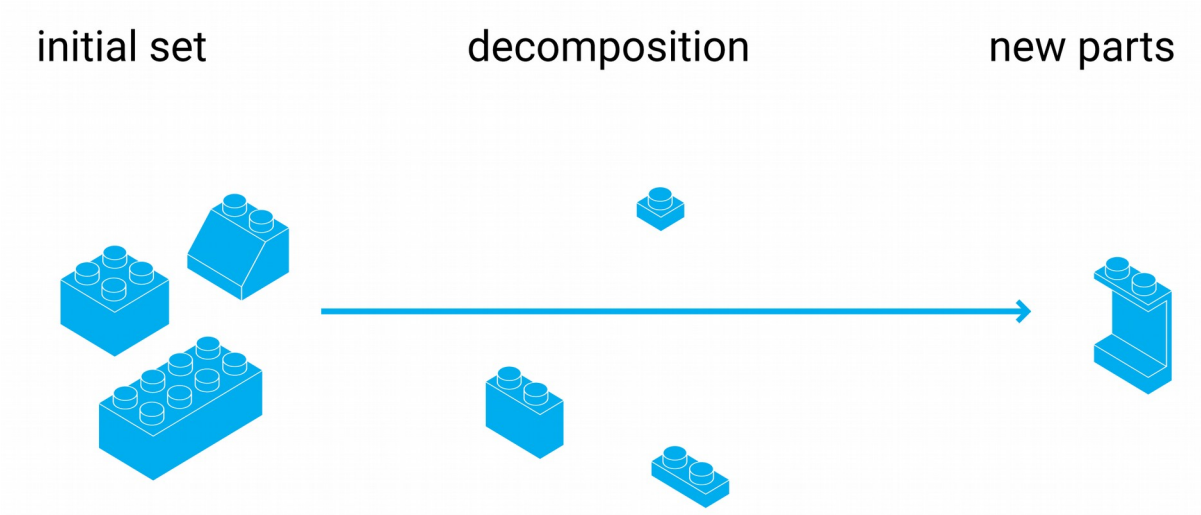


Figure 18: Additional example of emerging components

It must be understood that this process can alter the characteristics of the initial set of components, when they are built back from the resulting base components. This depends upon what parts of the final components are required to remain. Following the example of the transportation systems, the smaller set of resulting pieces, can only be achieved by overlooking factors such as appearance and energy-efficiency of the resulting transportation machines. The same phenomenon is exemplified with the playing card graphic icons in Fig. 19.

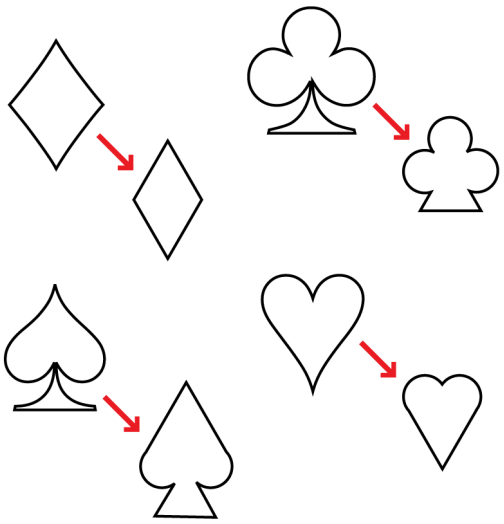


Figure 19: Example of changes in the initial components after the operation

Another caveat to the process, is that each component could have user-defined properties which change properties of the object. In this way, the process can be cheated in a way that the result is just a single component that has so many configuration options, that it can cover any functionality. In the example of the transportation systems set, it would be like defining a block of metal as the base component, because it can be machined and moulded in any way to generate different components. Here the designer's common sense must take a stance on how adaptable each component needs to be, according to the desired context of application. For some cases it does make sense that a component changes role by using a user-defined parameter: for example a bolt-nut component has the user parameter of how many turns to screw a bolt, which is a perfectly reasonable user-defined parameter, while allowing a wide gamut of configurations. For the current case of musical system design with an eventual application to physical units that integrate many of these components, there is a limit on tweak-ability, and the scripts that define their behaviours should be simple, and as monolithic as possible avoiding an excess of user input interfaces, or switch statements, for example.

The idea of molecular composition, as introduced in the *Brocs* and *Licog* explorations, was an interesting starting point, although they needed to be re-defined in many aspects for the purpose of this project. It is worth exploring an environment for molecular composition, based on the idea that an environment that can handle the musical *molecules* will also be able to sustain any other, more complex modules. Additionally, the molecular paradigms explored in the aforementioned experiences were very limited in terms that the environment was specified only for *global* musical events, meaning that resulting musical events could not be altered once emitted, but would take effect instantly, as if each component of a composition would have its own speaker. A different environment logic is required to build a modular environment with endless possibilities in the same fashion as a modular synthesizer, thus allowing *divergence*. Specifically, the best way to re-define the *molecular environment* would be to proceed with a *buildification* process using as the initial

set of components a *mono-sequencer*, an event-mapper, an arpeggiator and a *Licog*.

With respect to the communication protocol, if there is anything quaint on the way that a device is triggered, or about what a device outputs, it would compromise the versatility and compatibility of future devices. A good illustration, as always, is the Lego building block. By good luck or by a good decision, Lego has been able to keep innovating and creating new pieces, and allowing the user to build a very wide range of things, while still keeping compatibility with their earliest pieces. This quality depends on that very first design of the mechanical joint that the first *Lego* block had. To apply the *buildification* process in this prototype, a new modular environment was simulated in javascript. To this environment, a sequencer and a Licog modules where programmed and instanced in a way that it was possible to use them in connection. For each of the simulated components, its procedures were analysed as features or sub-components, in order to merge or split them into different functional units, or modules. According to the explained *buildification* process, the intention is to have the minimum possible amount of different components that would allow building the maximum amount of the initial set of components. It was expected that from exploding these two components, it would later be possible to build other types of systems such as arpeggiators, harmonizers, event mappers and so on.

To encourage modularity as suggested in the last exploration a global clock was not any longer used (Fig. 20, a). This out-ruled the Licog modules as they relied on the global clock. However, this opens the question of how clock-bound (e.g., a sequencer) modules could be triggered in an environment that is exclusively modular. In Pure-Data environment, any signal that is sent also serves as a bang which determines when to propagate the messages. This leads to a frequent need for modules to have several different outputs and specific operations. If there is a need for a module to wait for a clock signal in order to propagate, a specially dedicated module or additional inputs on each module would be needed (Fig. 20, b). As the intention was to homogenize the pieces and communication methods to the minimum, it was defined that instead, a message contains a header number which can be interpreted by each module depending on the module's functions. In this way, the distribution of the clocks becomes modular, with no requirement for a global clock bus. This allowed the existence of clock messages as distinct from musical event-messages, and therefore the connection between modules can be reduced to as low as one input and one output, while still allowing several functions (Fig. 20, c).

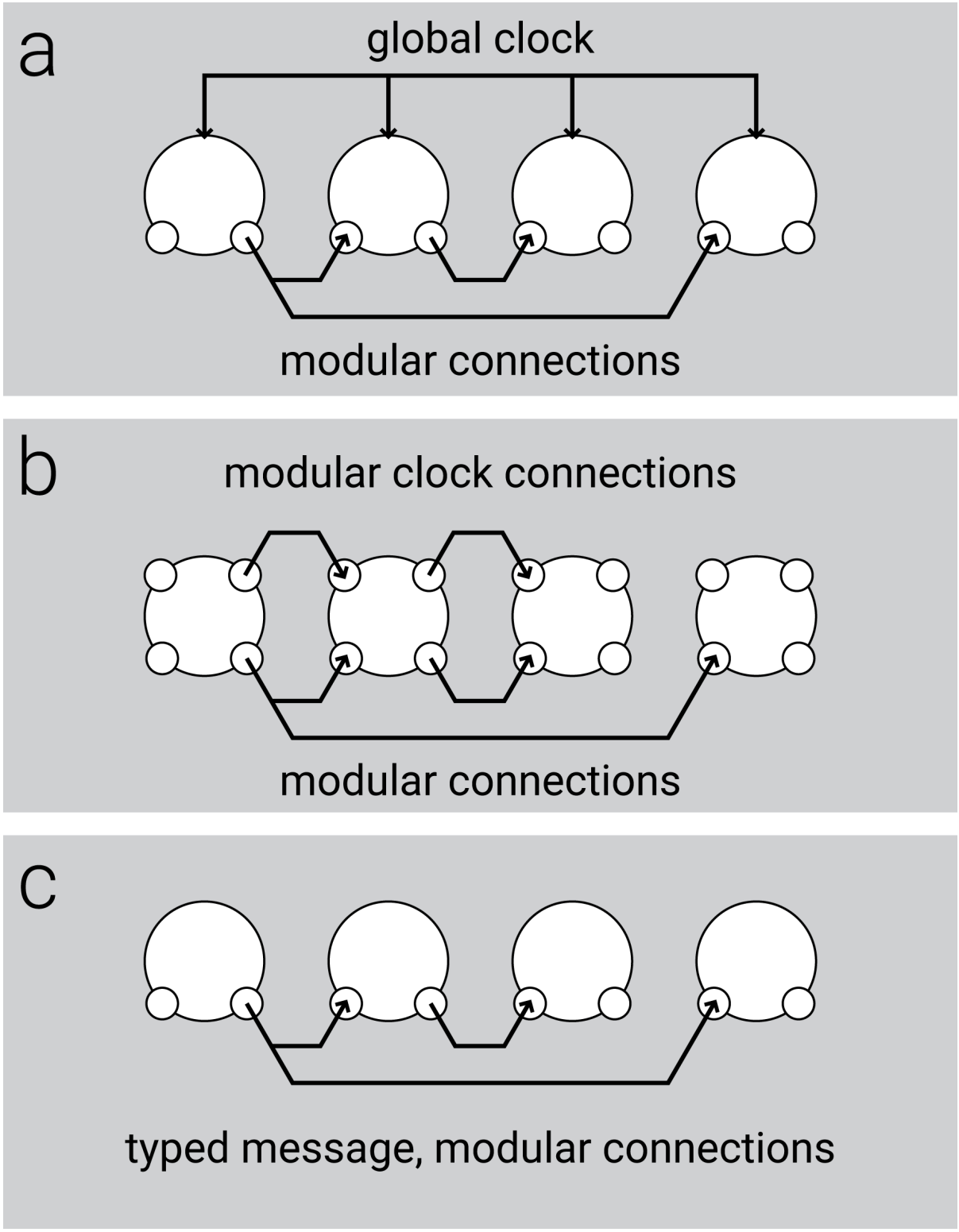


Figure 20: Three approaches to distribute a clock signal in a network of musical devices

This idea was later reinforced by the modelling of a FIFO module, which also needed distinct functions of *store message* and *send buffer* messages. If the functions were indistinct, it would not be possible to delay messages as it is required to make a counter, and to make a Licog module.

Given that the messages are typed and not dependent of an input, the clock message becomes a generic trigger message or *bang* which could have been originated in any other way. More interestingly, it would be possible for modules to manipulate a trigger signal into other type of signal by simply altering its values. A simple module was devised which would record any received message, except for a clock message. If a clock message is received, the currently recorded message would be propagated to the next modules. This facilitates the creation of memory and delayed triggers. This module later derived into a module which could hold any number of event-messages that could be triggered sequentially, in a *first in first out* (FIFO) fashion.

One of the most obvious modules, which was modelled at the beginning of the experimentation was a module that could send a digital signal to all of its outputs, once it received any signal on its input. After the idea that clock signals were mere messages that were interpreted as clocks by a module, it was defined that this signal generator module could actually be a signal modification module. This module could transform a trigger event-message into a musical event-message or any other. The module effectively operates one input signal for it to become another output signal. This module also could perform conditional operations as to define whether the message is propagated or not upon conditions.

One module that emerged and disappeared during the process was a multiplexer module. It was designed to send an incoming signal only to the output that is indicated in the signal itself. The utility of this module was replaced by the ability of operator modules to have conditional functions: by using many operators, it was possible to build the multiplexer.

During this process, most messages consisted of three bytes, making it potentially MIDI compatible. Two, however, were enough for the extent of this exploration: one number to select a function, and other number to set a value. Any additional numbers would serve to specify more in detail a theoretical note trigger. This led to the additional idea that messages could be of variable length, in which case the header could also integrate the definition of this length.

It was concluded that four modules could describe a wide range of composition elements such as a sequencer, an arpeggiator, a Licog element, and a harmonizer, between others.

- Input module: it converts any defined input into event-messages. its only parameter so far defines which stimuli triggers a bang. In the javascript prototype, so far, can only be either a clock pulse or the press of the space key. In a physical prototype it will probably be able to respond to hardware changes, and to any incoming MIDI or message signal.
- operator module: it performs one operation for each byte of the message. The operations can be arithmetic (e.g., adding one to the second byte of the incoming message) or boolean (e.g., propagate if a condition is true), making it effectively an input filter (e.g., the message passes only if the first byte is `0x80`). The operator calculates and propagates the input as soon as received.
- FIFO module: this module stores incoming bytes in an array, if the byte header | `0xF0` equals `0x20`, and sends + deletes the oldest byte of the byte header | `0xF0` equals `0x00`.

There are many other possible headers that may be implementing such as getting the message without removal, getting all the messages, getting the newest message or getting a specific message by index.

- Output module: converts bangs into output. Depending on the context, the output module may send a MIDI signal, trigger a CV, turn a light on or trigger a solenoid.

These modules would share a common, simple language of a string of integers where:

- first byte defines the function of the message and each module has a different set of reactions for each message header.
- There is a specific header for longer messages, and if the message has this header, the component must wait for a closure byte to stop reading the message. In such case, an escape character needs to be defined which takes effect in the context of long messages, so that sending any byte remains possible.

Event-message communications

within a program

[function, ... optional numbers]

on a serial:

[(Length<<8) | function, ... optional numbers]

specification of event-messages

This current idea of composition elements becomes similar to the implementation of Pure-Data, were modules can exchange discrete information, but in this case leaving away the continuous variables that Pure-Data handles such as audio buffers. This idea of getting a sub set of elements from the Pure-Data composition environment relates this project to Liam Goodcare's context sequencer (Goodacre 2018), which builds higher-level components by using Pure-Data. In this process, however the intention is to generate an

environment which is dedicated to live composition, which includes the patching of modules through a physical interface.

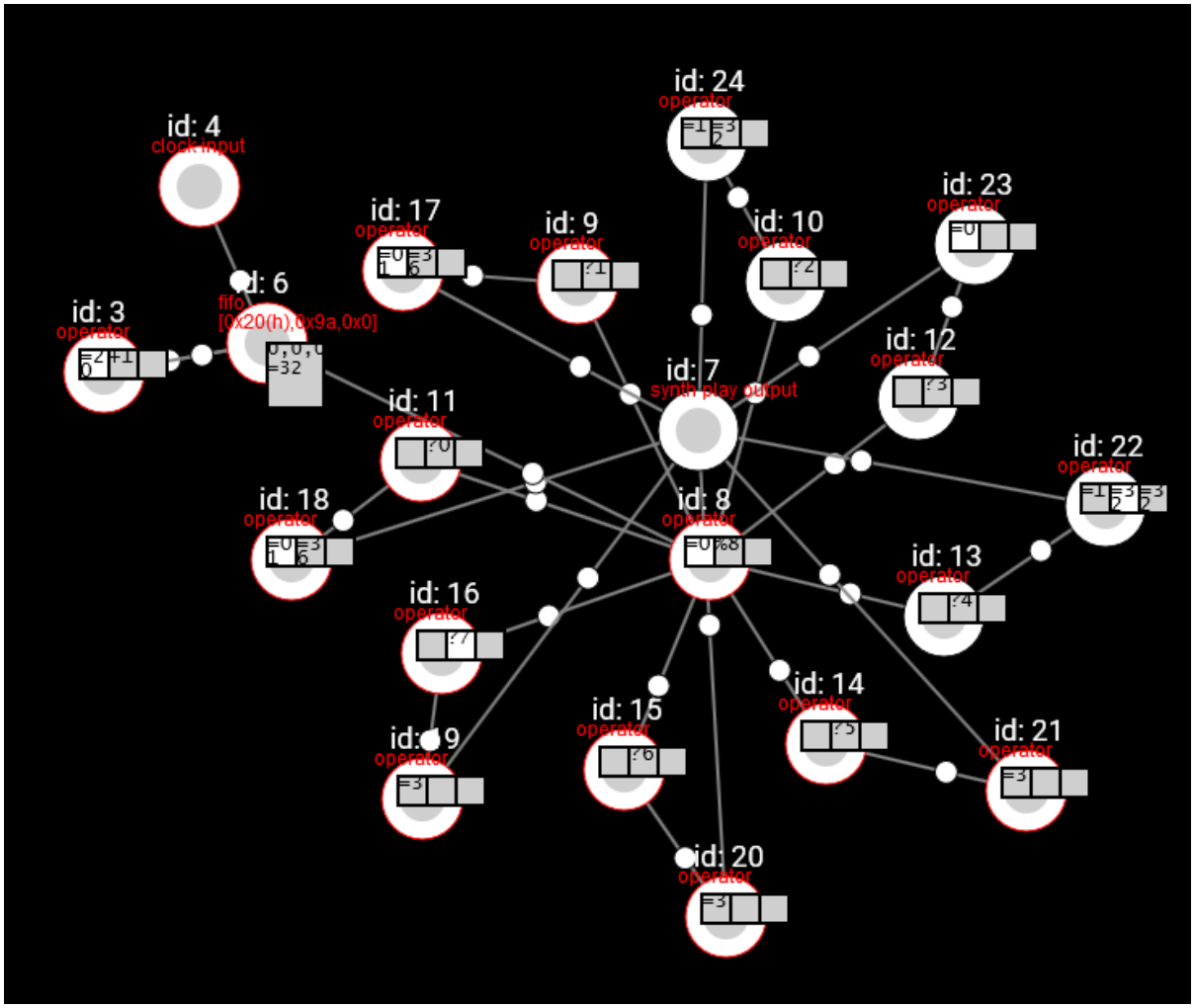


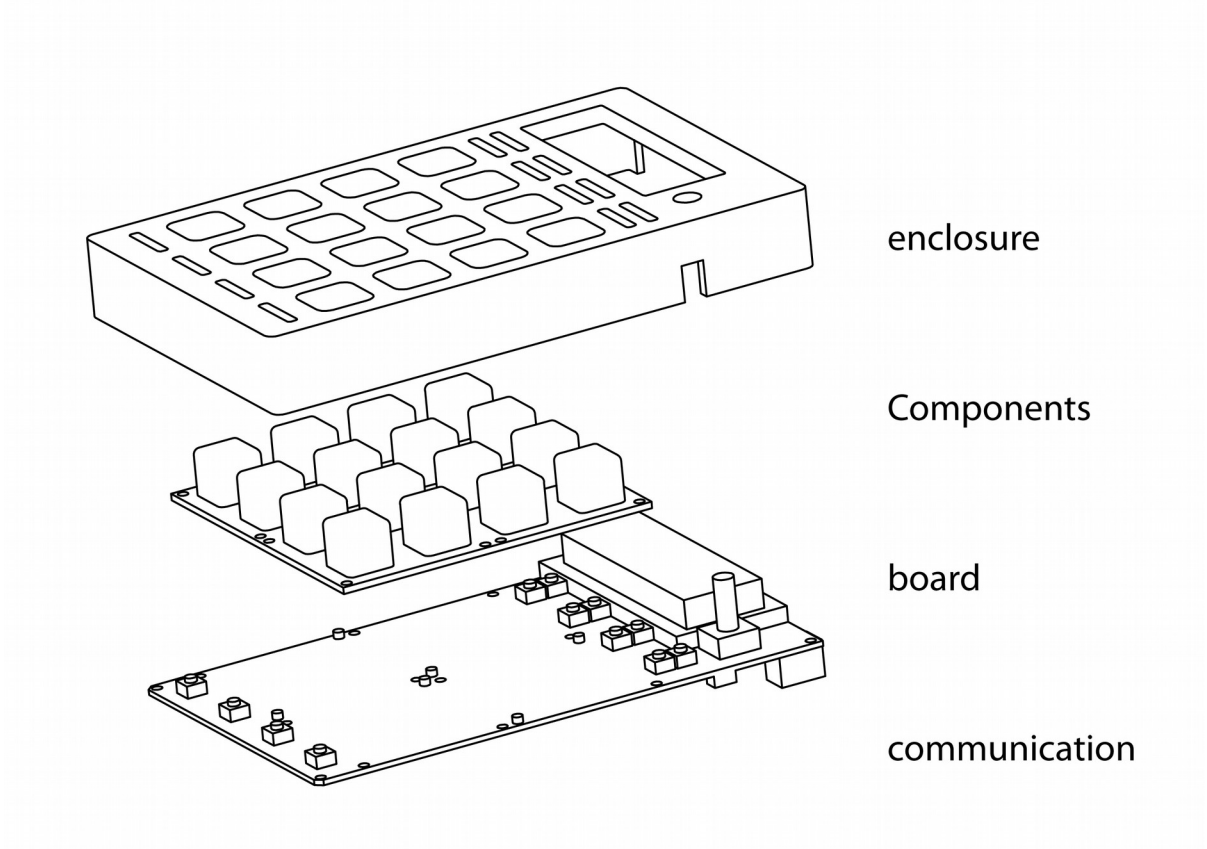
Figure 21: 16 steps sequencer

The Fig. 21 above shows how a 16 step sequencer can be made out of these components. *Licog* units are also easy to implement with these modules, as a signal can be stored in a FIFO until next clock, and send all messages in FIFO on every clock to the next *Licog*. It is also possible to build simple arpeggiators, scale mappers, and so forth. This definition of basic modules satisfactorily covers the domain required domain, although the definition of notes-off and control messages remain as an interesting future exploration.

Despite the idea of creating a set of hardware micro-operators that replicate this environment is very interesting, as a project it will be necessary to focus on more complex, and more user friendly ideas of a module. Modules built upon these modules are not easy to manoeuvre as the built entity: a built sequencer, for instance, would not be user friendly as presented in the picture, since changing the sequence length involves changing the structure of the system. It is also interesting to note, that given a definition of the environment that specifies the role of a module and the roles and characteristics of

the messages, future modules can also contain aberrations of the resulting basic units, without compromising the compatibility with the rest of the environment, as long as the inputs and the outputs belong to the same specification.

4.4 Development of Calculeitor



calculeitor design

When the development of the hardware started, there was no definition of a module, neither a proposition of making a modular environment. The interface would allow to explore the dynamics of performing with an environment that did not yet exist. The interface, thus, was based on the widely used 16 back-lit buttons matrix, resourcing to parts that are available in the market: a micro-controller, 16 button silicone keypad a screen, encoder, and tact switches. The decision to work with 16 buttons intended to facilitate the design process, taking into account factors such as costs and time involved in the design of a mechanical interface. This prototype herein is named x16, which reflects the 16 RGB LED's which uses. The design of a standard interface allowed early on to experience performing music with different interaction modes. It also allowed to measure the capacity of different micro-controllers. Depending on how the environment evolved, there would be a requirement to modify the interface to allow these differences.

The very first prototype of this device was attempted using a Teensy, because of its music-related capabilities and better processor. Teensy was discarded mainly because of the realization that the platform is not open-source, and that the libraries were adapted to the Arduino language in very inconvenient ways; specially regarding the pin address mappings. The immediate following step was using an Arduino *pro-mini*, and expanding the number of pins by using multiplexers, and using Sparkfun components for an easier build. In order to map the limited amount of pins of the AtMega328 to the large amount of pins, a multiplexer was added. This design was devised throughout many iterative sketches, of which the Fig. 22 is an example. The amount of connections required for this prototype caused inconsistent behaviours in a prototyping board (Fig. 23), because of the complexity. A printed board was designed in KiCad and requested from a board manufacturing service, in order to have a prototype of consistent behaviour.

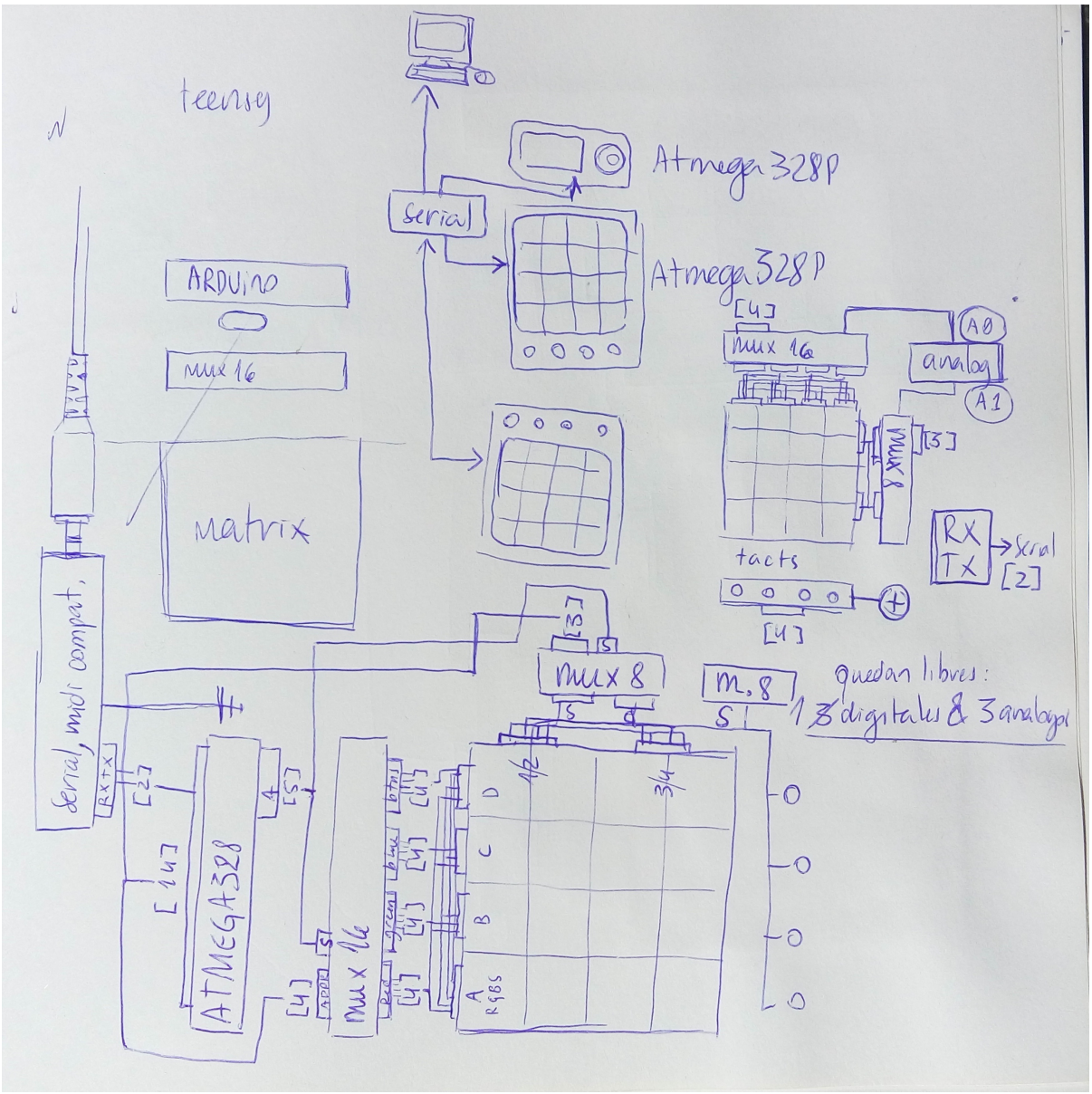


Figure 22: Sketch of the x16 prototype electronic design

While programming the firmware of this first version of the board, some of the limitations of the hardware became clear. The Atmel Mega 328 processor has insufficient memory to handle all the necessary processes, and the programming of the firmware takes too long time as to use it as a medium to develop the environment. Also the user interface LED's were too dimly lit and thus were not noticeable in places with strong lighting. This happened because, being behind a multiplexer, the micro-controller needed to scan through each led pin to create a persistence of view effect. It was decided to use these as controllers to access the computer simulation of the modular environment, and start developing a more powerful version of the same device with an Atmel Mega 2560 which would allow truly persistent LED's, multi serial ports, and more program memory.

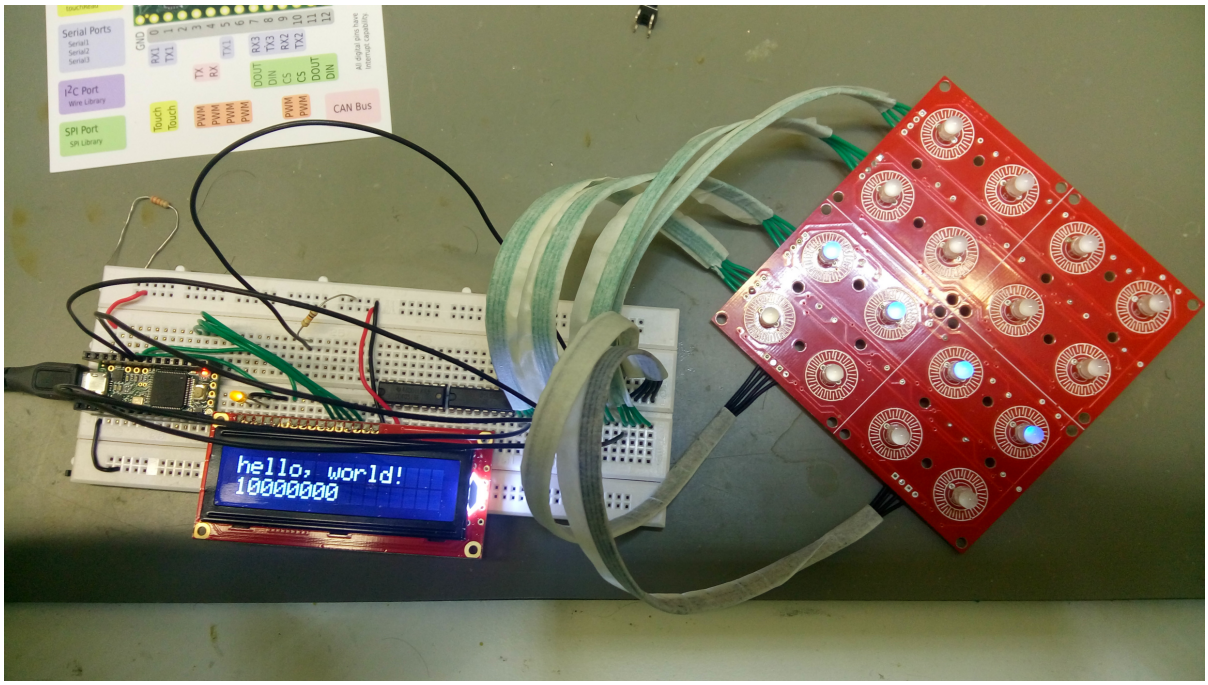


Figure 23: development of the electronic schematic for the x16 prototype

The enclosure in this version, thus, was designed with enough space to host a Raspberry Pi, which served as the main processor. The Raspberry was connected via serial with the custom board, and it was programmed to run a Node-Js service upon boot, allowing the device to start without requiring screen or SSH access. Devices could be connected using MIDI over the Raspberry's USB ports, which are detected when the Node-Js program starts. The Fig. 24 is a picture of one design sketch that was used to design this enclosure, and the Fig. 25 is a picture of the result.

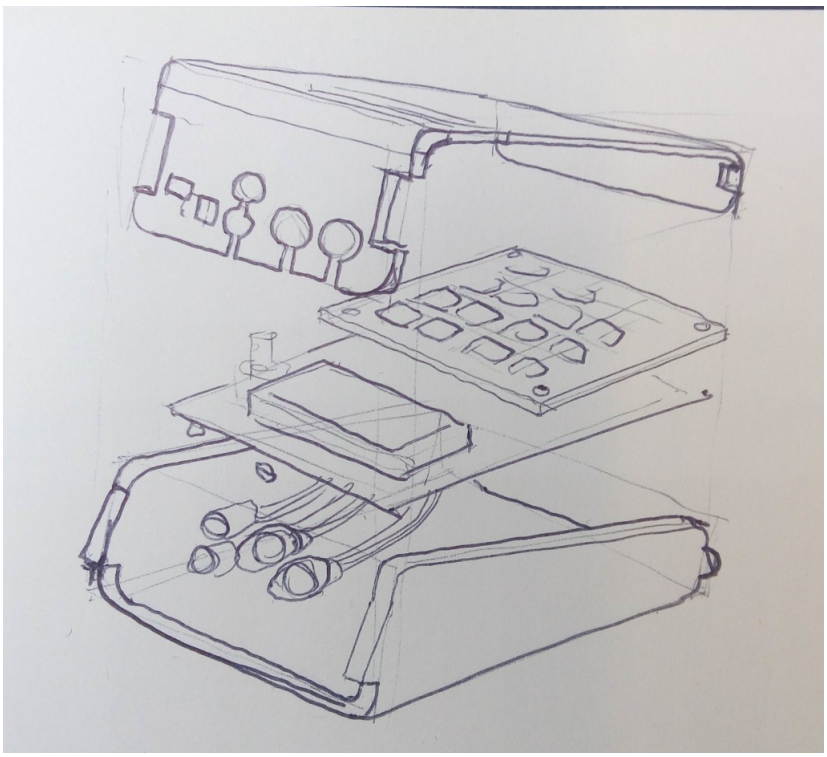


Figure 24: x16 enclosure design sketch

Figure 25: x16 calculeitor prototype

The following version of the board addressed the most important limitations observed over the last prototype. It was designed with 28 buttons instead of the 20 buttons previous versions had, each of which had an RGB LED (hence the name x28). The most important improvement was the processor program memory available, the amount of serial ports, and the intensity of the led lights. LED's needed not to rely on the persistence of view any more thanks to the WS2812 component, which possess a dedicated controller to store each colour value. The first prototypes and the board design were developed in Kyushu University (the Fig. 26 is a picture of one development prototype for this version). Along with this new version of the board, the Virtual-Modular environment was modified to be compatible with both boards.

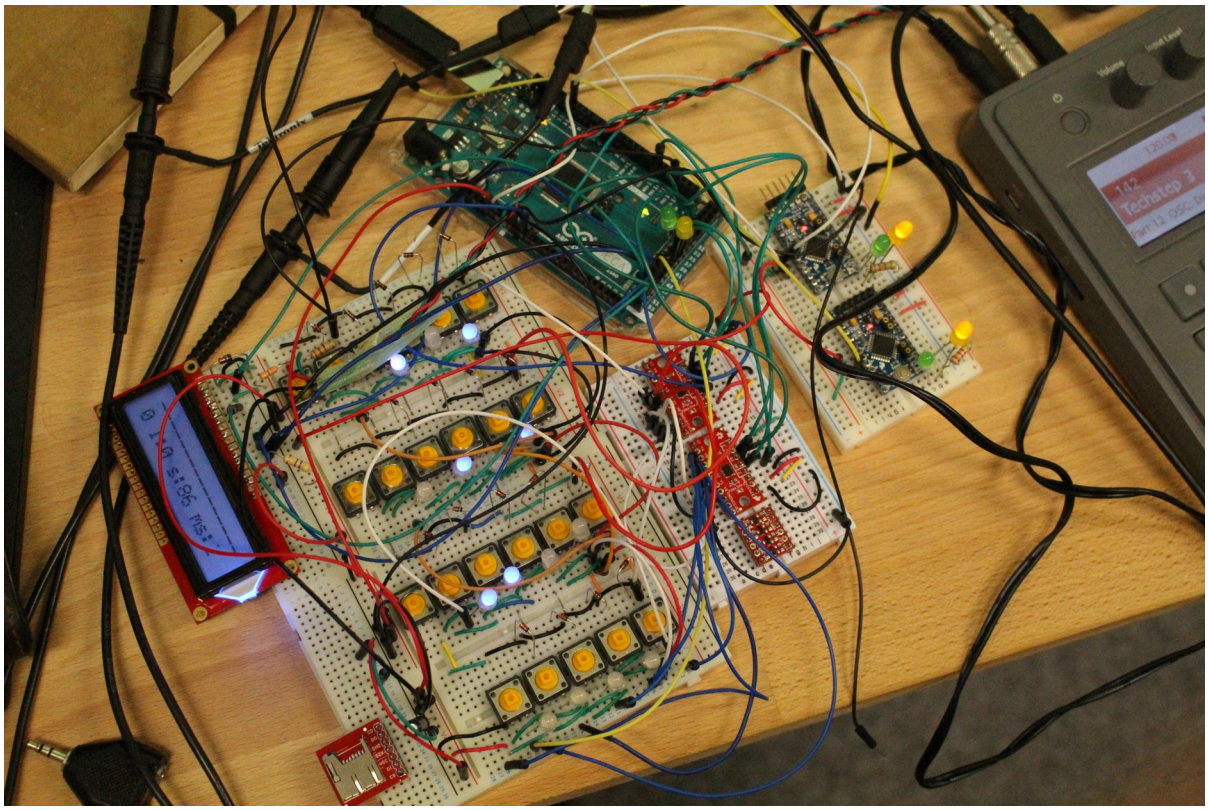


Figure 26: Development of the electronic schematic for the x28 prototype

The enclosure of this version were first thought as build from acrylic sheets, because of the accessibility of laser cutting. Later it was realized that bending and manufacturing in acrylic takes more labour per part than silicon casting. Many different approaches were modelled in parametric cad software, to speculate in detail about the cheapest and most ecological options available.

4.4.1 Networks

The design process of the networks happened in recurrence with the design of the environment because the design and interaction of the product is also involved in the mode of communication between nodes. An example of this is that if the network is point

to point, then the expected interaction of the user involves patching the modules in the same way as modules are patched in a Euro-rack system, mechanically. Otherwise, in a common bus network for instance, the user would be expected to virtually patch modules, as they are all already fully connected from the start.

The modular environment, in terms of network need a different set of terms since the roles of a hardware piece need not to necessarily correspond in a one-to-one relation with the parts of the environment. The first networking term, *topology* refers how a network electrically connects different devices to form a network. Also, an item that has a distinct identity in a network will be referred to as *node* instead of *module*. Both, although in some cases are, they *are not necessarily* the same entity. For instance, the crucial need to communicate several electronic devices, is not necessarily parallel to the need to connect different modules: a module can be sharing a networking device, in which case more than one module can be represented by a single network node from the point of view of the network. One real life example of the device versus node difference is how a single computer could represent two different IP's in a TCP-IP protocol, and vice versa. In this case, a computer could represent two clients while still being one single node in terms of network.

The main challenge in inter-micro-controller communication is to create an algorithm to prevent data collision. Data collision is when two nodes need to send a message at the same time. A bus cannot support more than one message at a given time, and a micro-controller cannot (or has a limited capacity to) listen to more than one incoming message. This is similar to spoken communications, where it is not possible to listen more than one person speaking at the same time.

There are other important factors to take into consideration when designing the communication, the most predominant being the achievable data rate, because this determines the amount of interaction that will be possible between units. The reliability of the network is also crucial. The ratio of information that is lost against the total information sent can be divided to form a data loss ratio. It should be 1 or very near. Information can get lost mainly because the reception device may be busy, because the message was destroyed due to noise, or because messages from two nodes took place at the same time in a shared communication line.

The previously mentioned factors are in tension with processing that is required from each unit in the network, because the units need to do other things than only communicate. If a network requires a highly active participation from each node, the availability of the processor for other tasks will be reduced, increasing the processor power requirement for each node.

Regarding to the topology of the network, the directionality is important: many networks work in a paradigm of master to slave, which most often is implemented to a hardware level. For instance, most of master to slaves network are connected using two buses, one where slaves use to communicate to the master and other for the master to communicate with the slaves. Networks which do not work under this topology, most commonly being one-to-one, meaning that only two nodes can participate at any given time Fig. 27. For a

network that intends to allow a heterogeneous interconnectivity, the most obvious scheme is point-to-point. Under some circumstances, however, the ability to communicate more than one node is a desirable feature of master to slave networks.

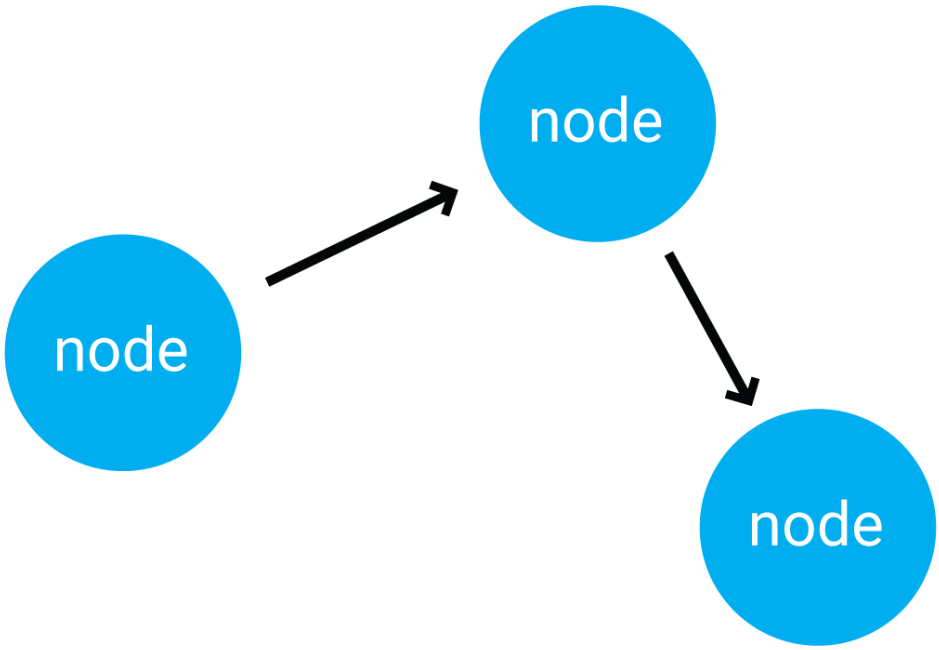


Figure 27: scheme of point-to-point networks

The required network also needs the ability of hot swapping: consists on being able add and remove nodes to and from the network without disrupting the communications or having the new node ignored. This is necessary since the environment is intended as modular and thus needs to afford re-routing of the communications to change the composition system. Hot swapping, however can be virtualized in cases of bus networks, as it will be explained later.

The modular network design starts with a careful consideration of different communication options and their characteristics. The following step consisted on iteration over different specific protocol ideas. These were sketched in detailed drawings of the topology and the decision trees. One of these sketches is shown in Fig. 28. Each topology can be more or less challenging in different aspects. It can be generalized, however, that for each possibility, different cases were put into test by following the decision trees and their expected effects, checking if the resulting device states could produce locked states or data collisions. The most important cases taken into consideration were: a) a device is powered up alone and then an additional device is connected, b) many devices that are already connected are powered up all at once, c) one device is disconnected from the

network at run time, d) an event causes a communication line to attain noise. From each one of these events the network needs be able to recover and keep the communications. For some protocols, specially the common bus based network cases, many iterations needed to be performed in order to strategically define a decision tree that does not get locked at any state.

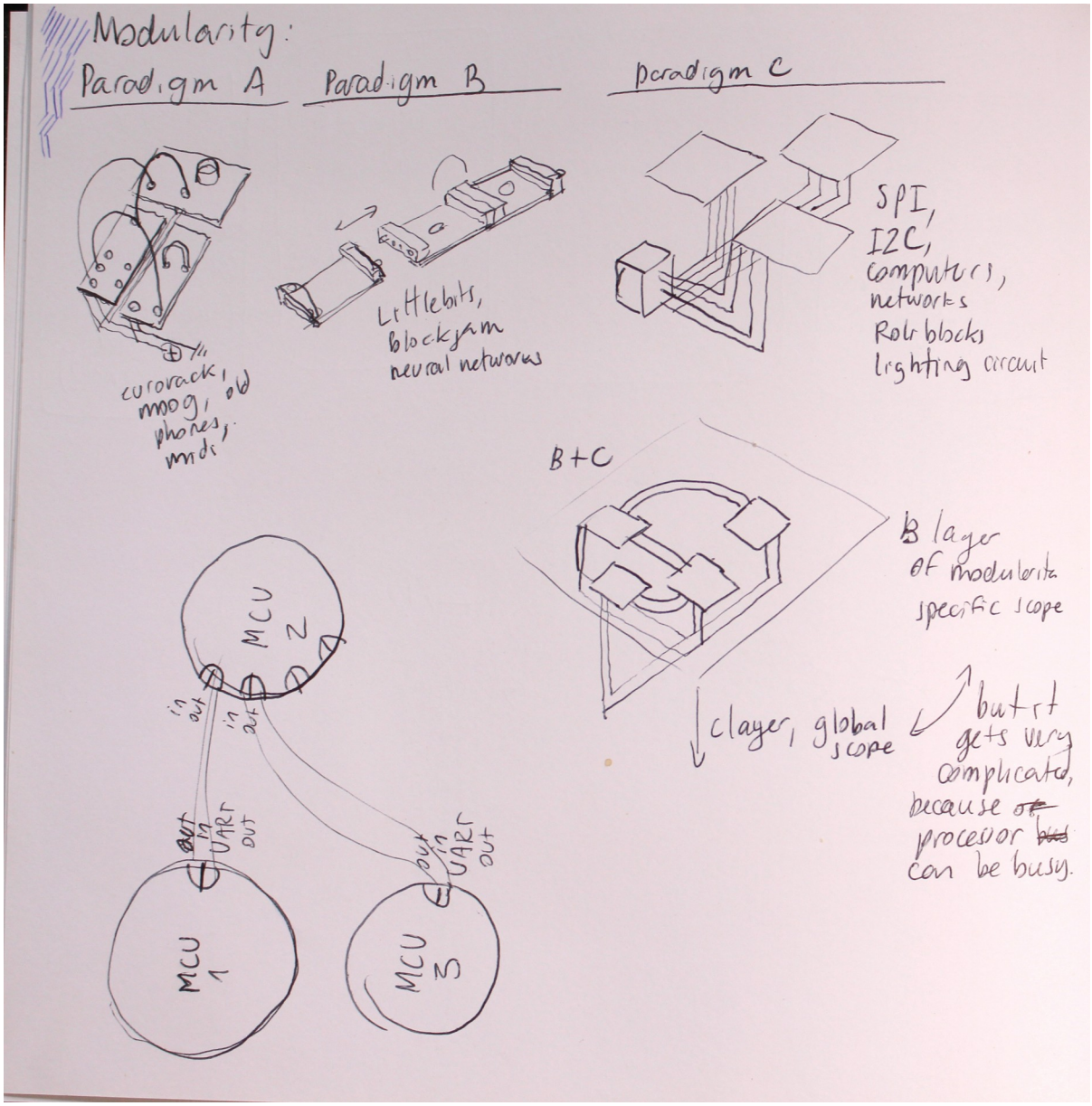


Figure 28: Picture of one of the sketches where network types and topologies were brainstormed

The first and primary type of possible network is point-to-point. The idea of the point-to-point network is that each node is only aware of those nodes whose inputs are connected to it. Other example is the software Pure-Data. The most common point-to-point communication standard is the RS232, which is similar to MIDI. The challenge with RS232 is that a unit may need to receive signals from more than just one other unit and point-to-point networks require one dedicated transceiver for each input or output. AtMega2560

luckily has four RS232 pair of pins that could permit this use. TCP-IP is another protocol capable of point-to-point topology, which interestingly are used to communicate between the well known Pioneer CDJ turntables. TCP-IP protocol was found, however to require high level implementation that would discourage the use of low level micro-controllers for specific use modules.

One idea to extend RS232 to attain multi-input capability is to use a multiplexer. An RS232 reception pin (RX) would be connected sequentially to different multiplexer pins, theoretically allowing any quantity of outputs to a single port Fig. 29. This idea could work if the system has other, parallel multiplexer that distributes to the sending devices, an electric flag[10] granting permission to transmit, as a consequence of the multiplexer being connected. This idea herein is referred to as *polite serial* since it is Serial RS232 with the difference that the devices wait for their turn to emit signals.

10 Flag is a simple concept used in electronics, where a boolean type of information can be communicated or stored by using a voltage or its absence as indication of the *true* or *false* value of something.

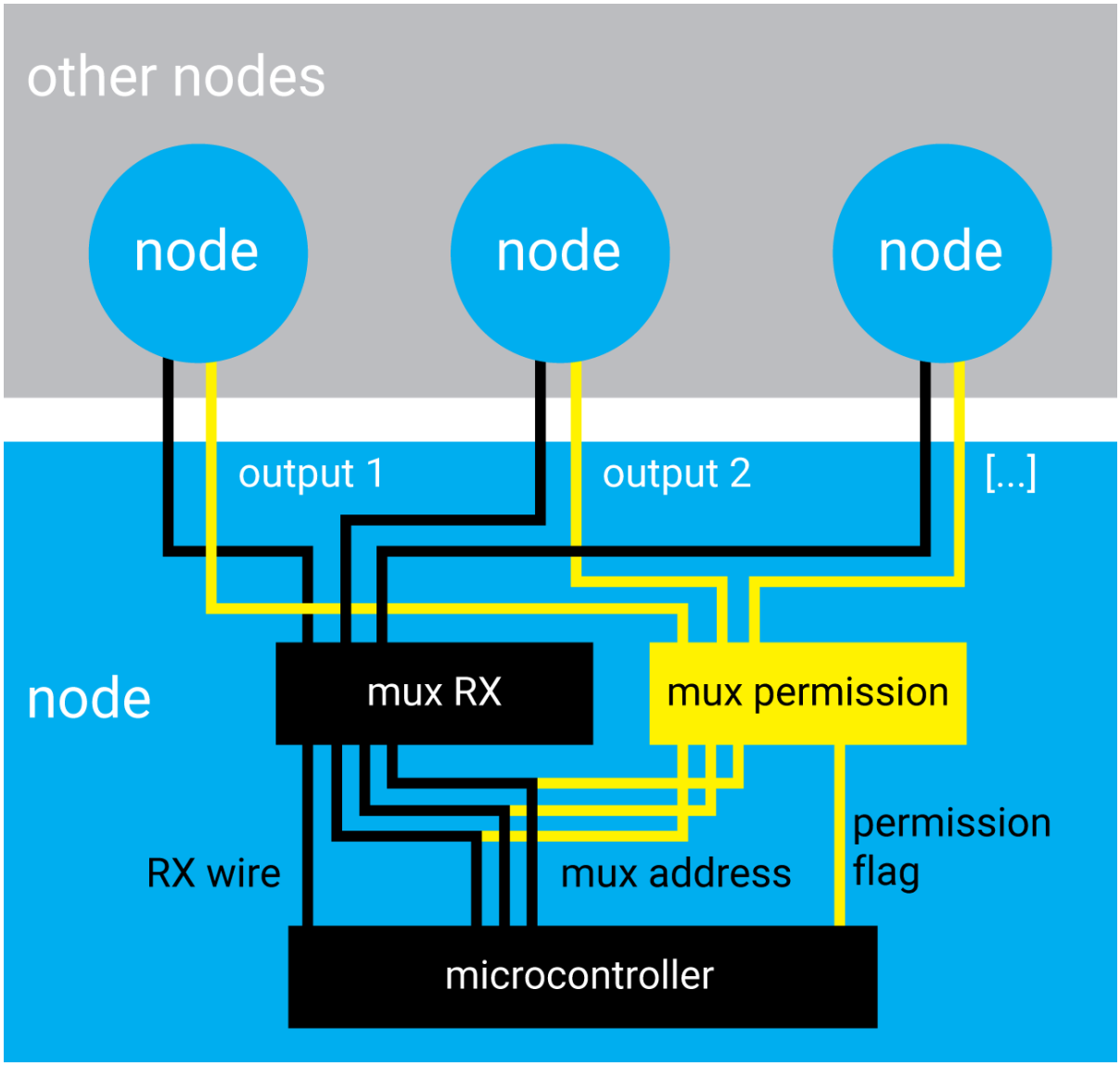


Figure 29: scheme of the multiplexed input to form a polite serial network. The TX wires can be branched without multiplexer.

Another protocol explored, also based on the RS232 was focused on minimizing the required amount of physical serial ports: to run this protocol, similar to *polite serial*, it would be required to use a serial input (RX), output (TX) and a digital pin. It was based on the idea of a token bus, but having a component that registers an address for each module that is connected in the network. It was inspired in the Token bus and I2c, thus it was named *token bus homogeneous network* or, in short, TBHN. The concept is the same as in a token ring, only that in this case, there is a token line, and to there is a module in charge of restarting the token every time it reaches the end.

A shared bus network consists of a single bus to which all nodes communicate Fig. 30. Two advantages of a shared bus network are the ability to monitor the whole network by monitoring a single wire, and the possibility of optimizing the flow of events for lower latency. There are two drawbacks: one is that each node gets a portion of the bandwidth

that is in inverse proportion to the amount of nodes in the network (whereas in the case of distributed, each network has a different bandwidth). The other drawback is the loss of the physical interaction of plugging and unplugging terminals manually: given that every node is connected to every other node, what determines the messages to read from the rest, is determined by software. The connection between components, therefore becomes virtual.

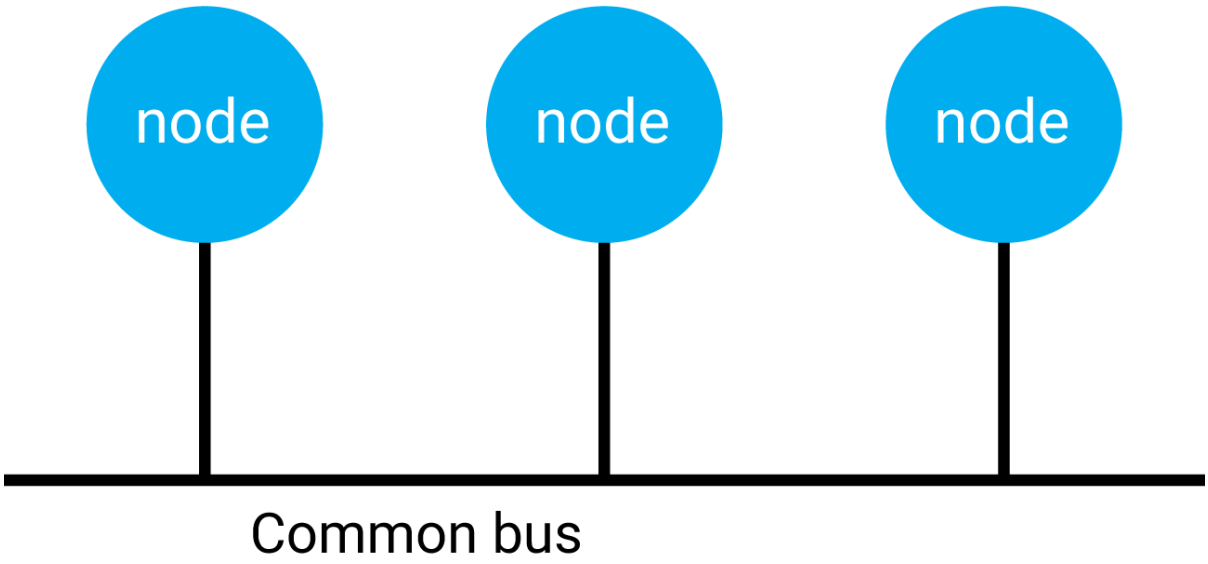


Figure 30: scheme of a common bus communication topology

It is also conceivable that each node's input and output is a common bus network, thus allowing the desired physical, cable based interconnectivity. This case can be exemplified with I2C Fig. 31: it would be a candidate; if it allowed direct slave-to-slave communications in a bus. However, each node could be thought as being both an I2c master and slave, being a master of its inputs and a slave of its outputs Fig. 32. In this way a protocol such as I2C can be turned into a point-to-point network. This same idea can be extrapolated to most other common bus protocols available.

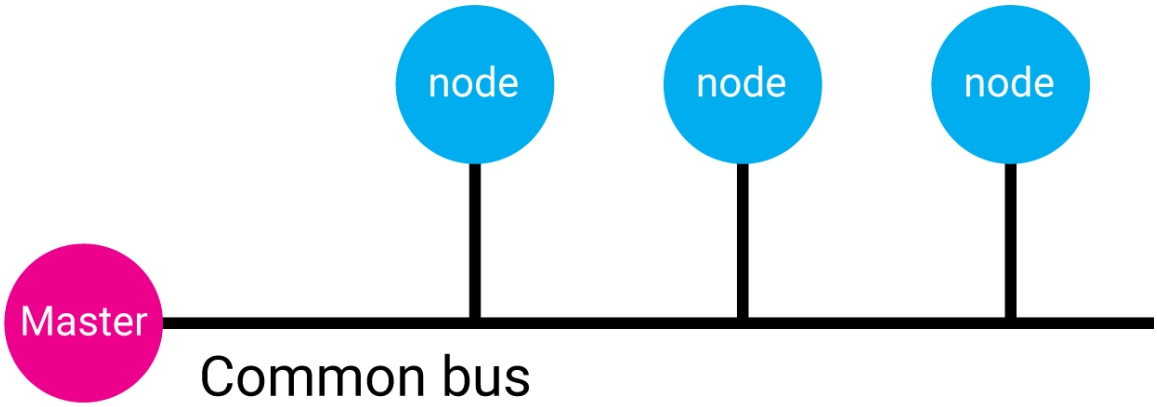


Figure 31: Example of a master-slave scheme on a common bus network topology, such as I2C

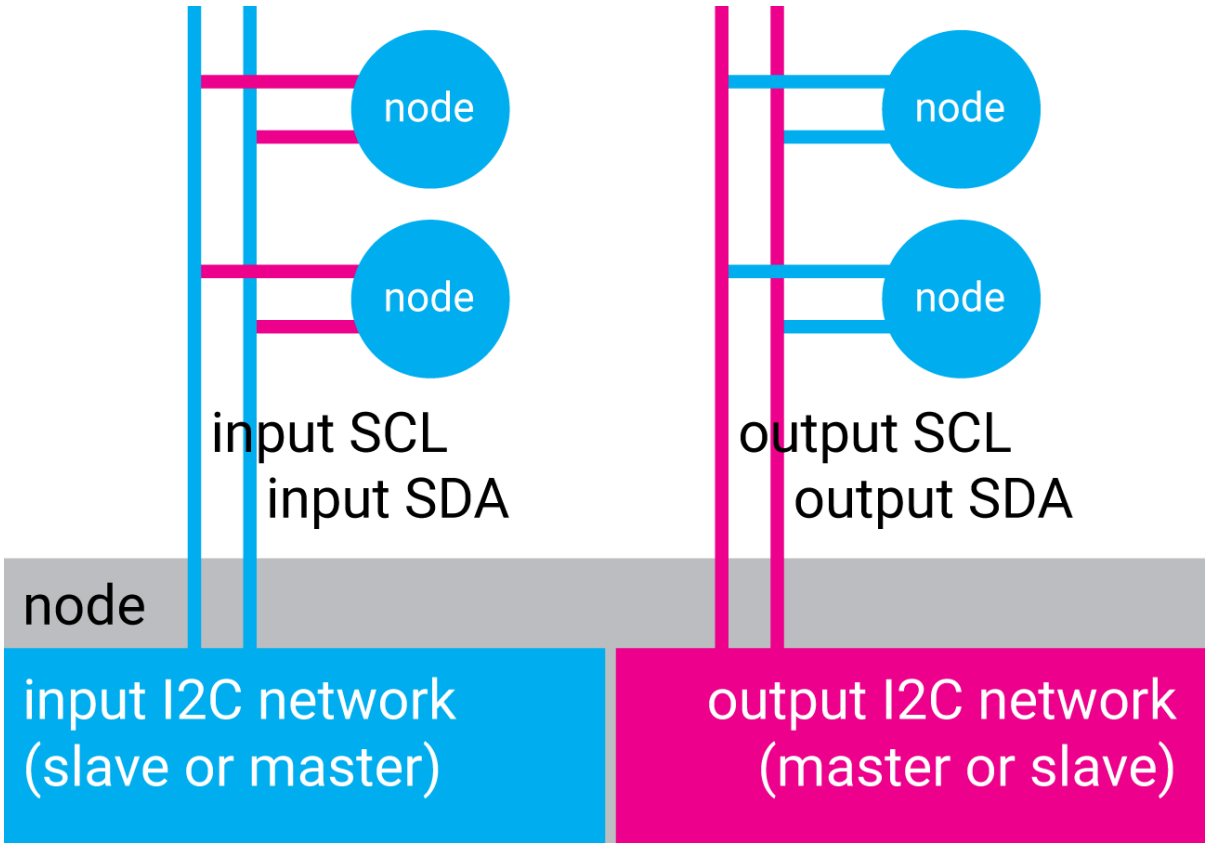


Figure 32: Application of a double master to slave communication scheme to produce a point-to-point network

A feasibility exploration of using a common bus network that allows direct node to node communication, a hybrid between Token ring and master-bus polling was designed and tested. The topology and scheme of this network is illustrated in Fig. 33. Token is an imaginary signal that is passed from one node to another, sequentially. Every node

acquires the right to write to the bus only when it has the token, and it can pass it to the next node once the writing is done. This is one of the fastest protocols to distribute writing permissions, since it does not require to use bus time for networking related information. All the bus time can be dedicated to payload messages. This system will be referred to as *token bus homogeneous network* or its abbreviation TBHN.

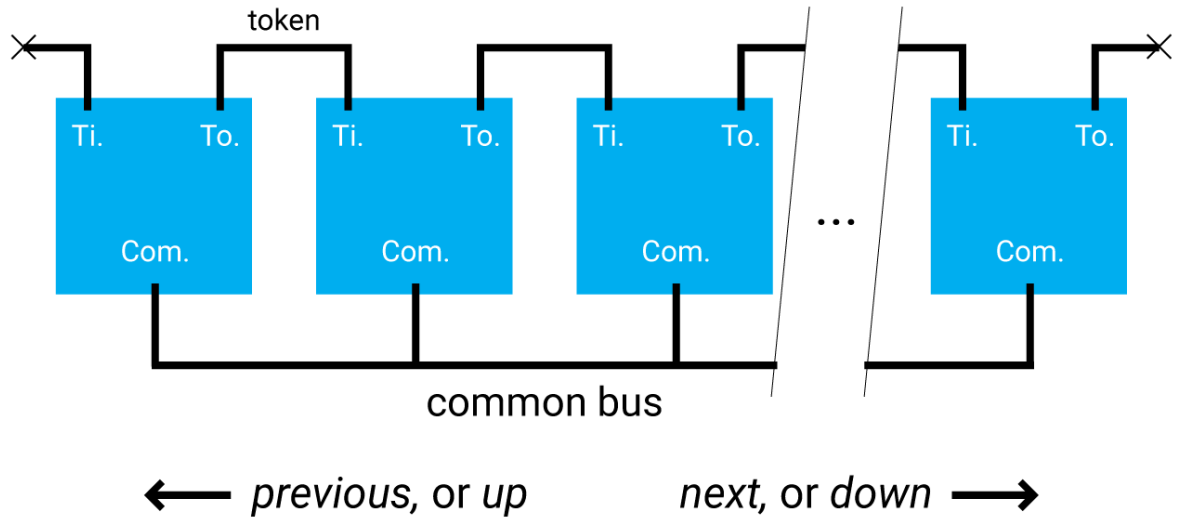


Figure 33: TBHN example

The network was set out to allow any node to broadcast information to all the other nodes directly, and redefine a Master at run time in case a master is unplugged, or not having a Master at all. Being a shared bus network, as mentioned before, the interconnectivity among nodes need to be determined by software rather than the physical connection. Instead the changing of connections among the devices would be software-based, it is necessary for the devices to keep track of addresses on the other devices.

In a practical sense, each node needs three pins dedicated to the network: a token input (TIP) pin and a token output (TOP) pin.[11] Also a common bus pin named COM, which reads and writes the serial bus. The pins and interconnections among nodes are represented as squares in the Fig. 33. The TIP and TOP pins are connected in chain from node to node, while the bus pins are connected to the same cable among all the modules. In this implementation the bus is different from an RS232 in that both read and write pins

11 In theory a single pin could fulfil both TIP and TOP functions.

are connected to the same cable, as opposed to RS232 which use one for each function. This is the basic fact that allows any module to communicate with any other module.

The protocol was defined under a set of basic rules. First, each module has three states, consisting of Listening, Broadcasting and Connecting. While in Listening state any normal node is on high impedance mode, reading the serial bus. On Listening state, the node is also reading the TIP pin. If TIP pin is set to 1, it switches to Broadcasting state. On Broadcasting state, the node sends all its available information if any, otherwise a "no information" header. Finishing this, it turns the TOP pin to 1, causing the following node in line to switch into Broadcasting state. When any data is received, a node turns the TOP back to 0 if it was set to 1.

Every node has a unique ID, starting from 0. The node number 0 is special role. When a node has just been powered up, it starts in a connecting state. On connecting state, the node has no address, and has the TOP on state 0. It is listening to all the broadcasts, and keeps track of the highest address in the network. When its own TIP goes high, it sets its own address to the highest + 1, writes a connected message to the bus, and sets its TOP to 1. This should cause a chain reaction where all the modules assign their addresses incrementally, if powered all at once. TIP pin is pulled to the state 1, causing the first in line to start the chain reaction from 0. A module can detect if there is a module before in the chain by detecting the voltage on its TIP pin. It is normally high, but a connected node sets this pin to low. When a node is on Connecting mode, if it detects 1 on its TIP, and has not registered any address, it means that there is no lower module. It will set its own address to zero.

A message in the TBHN needs to contain some bytes that indicate the functions of the message: origin and header. An origin byte represents a unique identifier that indicates what module has broadcasted that message. This unique byte is registered on each other module which intends to read messages coming from that module. The header byte indicates the mode of the upcoming data. It is divided in two nibbles: first nibble indicates the mode of the message (broadcasted, addressed, empty, offline). The second nibble indicates the length of the upcoming message in words.[12] The maximum message length is 16*4 bytes. If the second nibble is F, the receiving modules will wait for a special termination byte. Depending on whether the network is normal low or normal high, an address or header `0X00` or `0xff` respectively should not be used since these are equal to bus silence, and can account for devices that got disconnected or failed at run time.

Three devices in the network broadcasting messages, that were also sent to the laptop's serial port. One of these devices is a prototype of the x28 board. The signals were monitored in a digital oscilloscope, and also translated into MIDI to provide a perceptible sense of rate.

In order to implement this networking protocol, a set of steps were designed and followed. The design of implementation steps helped a gradual implementation of the network in such a way that it was possible to monitor with clarity all the critical aspects of the

12 A word in computer science is defined as four bytes.

network, delimiting the amount of possible errors to a range which is easy to manage. The step one consists on getting a common bus to work; meaning every device can read and write to and from the same cable. The challenge with Rs232 devices is that they have separate RX and TX pins, which do not necessarily work if they are connected together. Three Arduino boards were connected to the same bus and programmed as to be able to address each other individual node in the network by a hard coded address, and that each one could respond with their own ID plus a string. This is to prove that it is possible to use a hardware RX pin, which is turned into a TX pin by software, and there can be communications through that pin. It also allows to chose the best pin impedance mode for the listening state so that it does not interfere with other devices that might be writing.

The second step consists on achieving automatic address assignation. The Arduino boards are tied with the TI and TO connections, as expressed in Fig. 33. One single Arduino board was set to reflect in the serial all the signals that happen in the common bus. After the automatic address assignation, the Arduino board that is connected to the serial should be able to address individually each other board as in the previous step.

The third step of development is automatic token: the Arduino boards should start their activity without input from the node that is connected to the serial. The message length is fixed. The activity can be seen in the serial output of one of the nodes.

At the fourth step of development, there is a basic working TBHN. This steps consists in allowing node to be added or removed without compromising the network. In this case, the effect was granted automatically, because the continuity is given by the physical cable between nodes, thus removing a node becomes a complete removal from the network. After this steps many bugs emerged, related to which specific state was the network when the device closed or started communications. These bugs need individual addressing according to the specific transition that causes them.

According to the relation of this development to the design of the environment that will be described later, it was determined that TBHN development not to be necessary, however interesting for other purposes. The protocol at the stage that it was left allowed 43.5 messages of 8 bytes per second (the frequency of the TOP of one device was 14.5 Hz, having three devices on the bus) with a payload is 6 bytes per message. It was observed from the busy versus silent time in the bus, that there is room for duplicating this message rate. Additionally, it would be possible to raise the baud rate, allowing even higher data rates. Given this measurements, the latency down the token chain is very low, and up the token chain is around 10ms. There is a chance that an *end of line message* could be implemented, intended to be sent by a node that detects no following node. This *end of line message* would help the master to react instantly without waiting a timeout. The testing also suggested that the header byte should go before the origin byte and not after as initially specified. This reduces the bandwidth usage, because a module could refrain from sending a message by using a null header. Otherwise the origin byte becomes redundant for an empty message. This change of order theoretically would allow each node to host multiple virtual nodes that could be addressed by the network individually. If the following steps described at TBHN development repository guidelines, it would become an interesting communication protocol with a distinctive ability to perform slave

to slave communications.[13]

To this point, the communication protocol available to use are direct RS232 communication, *polite serial*, *TBHN* and point-to-point I2c. Despite the efforts put in the testing of TBHN, it was concluded that common bus networks are less suited for a modular environment than a point-to-point network. The most important point is the inverse proportionality between amount of connected modules and available bus time. There is also a common factor among common bus networks of high strain on the connected devices since they must constantly use processor time in order to participate in the network. During the initiation of TBHN experimentation, there was an idea of using a global scope of signals such as clock. The global scope obtains a bandwidth advantage from a common bus because a message needs to be sent only once to all the modules if it is global, while node to node messages have a disadvantage of sharing the available bus time. The deprecation of a global scope that was defined from the definition of environment (it took place at the same time) also removed the advantage of a shared bus. Furthermore, applying a point-to-point, bi-directional RS232 opens the possibility of later using *polite serial* or other similar technique in case a greater expansion of inputs is needed. The additional benefit of point-to-point networks is that the current micro-controller being used, the ATmega2560 possess 4 hardware full-duplex Rs232 ports, which makes the implementation easier at the current stage.

Among the study of the virtual implementation of the environment and the physical study of the device networking, a message type was specified which would be flexible enough for modular expressions, but also efficient in terms of bandwidth requirements. It is based on RS232 in the sense that messages possess directionality. This message needs to reflect one header byte which defines the role of this message among the available roles in that input, and a dynamically defined amount of following bytes which specify more information in detail, being the first ones, the most important, in a way that an event-message can be truncated to different extents, allowing messages to still work with different degrees of data loss.

It was defined from the testing, that it is interesting to use negative numbers for operations such as recording a subtraction into an operator, or automating a negative transposition of a melody. A negative point, however of a signed integers is their limited range. This is what caused MIDI to possess a range from 0 to 127 instead of 255. For the implementation of control messages in this environment, the number 0 represents the middle position of that parameter (instead of the minimum, as it was defined in MIDI). Control change event-messages can also be defined in higher resolution by using additional bytes to represent decimal points. Numbers with negative value can also be described by integers which are below 127. The conversion from unsigned integer to integer thus becomes `int = unsigned byte-127`. This produces a range of numbers which are asymmetric, where the maximum negative number is –127 and the maximum positive number is 128. The number 128 thus can be used to represent a transparent or undefined

13 There was an experimental implementation of this type of network, whose code and notes are detailed in the *Token Bus Homogeneous Network* repository, available at: https://github.com/autotel/TokenBusHomogeneousNetwork.

number, as to express event-message values whose values needs be filled with other default value.

4.5 Exploratory iteration in the Virtual-Modular environment

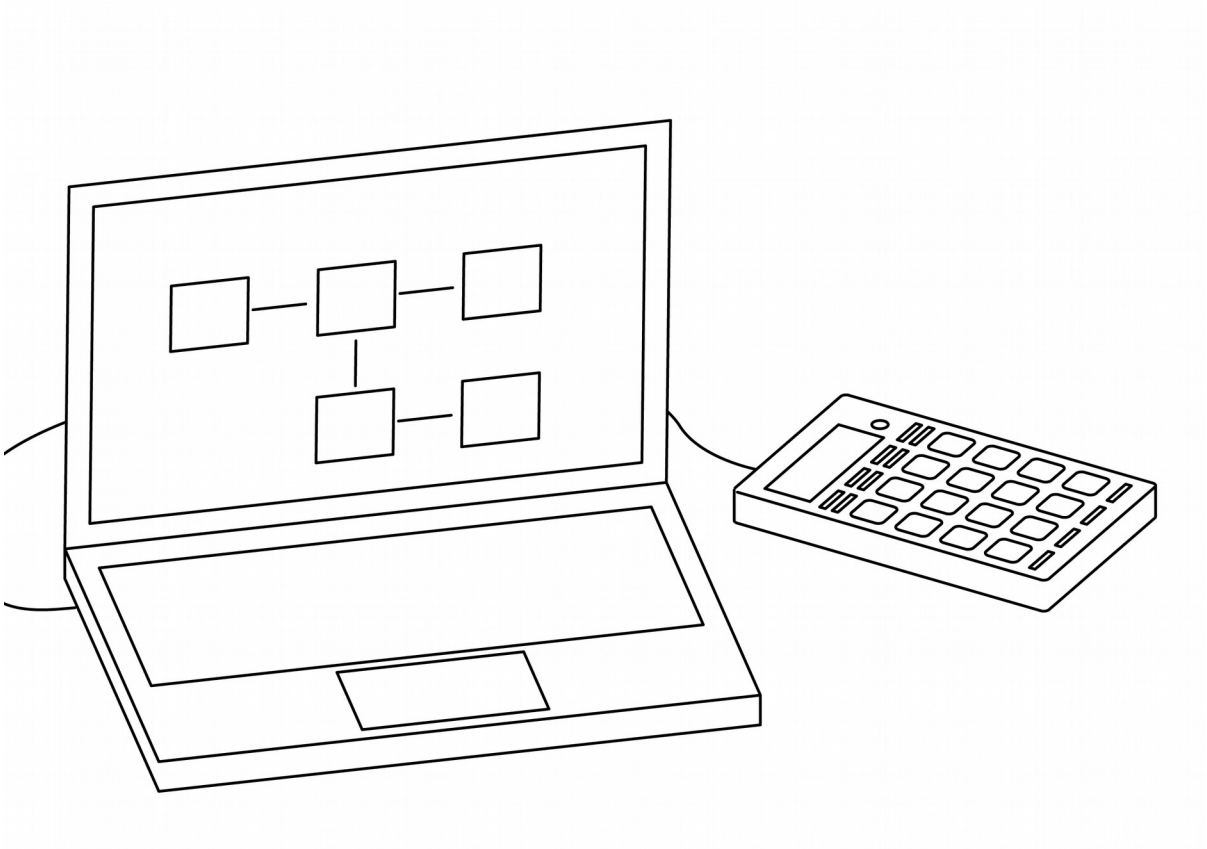


Figure 34: Schematic representation of a virtual implementation of a modular environment

The exploration of a virtual implementation of this modular environment was motivated by technical factors. The first version of the *Calculeitor* hardware was based on an *Atmel Mega 328*. The first attempts at implementing the sequencer functionality in this chip demonstrated that the hardware memory would not allow the intended composition features, but most importantly, it posed challenges on how to communicate with other devices in a modular way. Being a design project, and not an engineering project, the facilitation of a design process was prioritized. This is why this device was re-purposed as a controller which would be used to access a modular environment that is simulated in a javascript application, in a computer, as expressed in Fig. 34. This allowed a faster evolution toward the definition of a composition environment and gave place to a fast evolving, iterative exploration of this environment.

To do this design process without harnessing the possibilities of implementing it in the future, it was designed under a modular paradigm, where each module must be strictly isolated from one another, except in the cases where they have explicit connections, in which case these modules can only connect through event-message objects, which are representations of a message that can be sent via a serial. This limitation proved being useful beyond the hardware compatibility, it caused the interactions among modules to be clearer, and helped with the on-going definition of global meanings for each message's header.

The system was tested by doing test runs. There are four types of test runs, the most simple being code debugging, where the environment is run with the intention to make sure the system does not stop due to programming mistakes. The other test runs consisted on testing the environment with the intention of checking its playability. For each change that is done to any module, the environment is launched to assess whether the change grants the environment with a greater affordance of divergency and a better musical outcome, also checking what other outcomes become impossible once the change is implemented. The third type of test run consists on performing with the environment in public, to assess whether it is possible to lure an audience into dancing and keep an interesting progression of the patterns. It also proved useful to test the mental strain of using the environment in the stressful context of the live stage. Finally, there were test runs done with other users. These were done with people that had some level of experience in making of music, preferable with advanced production and programming of digital music, since its usage is not easily learnt.

Debugging test runs are often part of the design process. Sometimes a bug reveals that an imagined module or feature cannot exist, as it has assumptions that are either illogical, or require a code which is too complex. Complex code is not a problem in terms of being impossible to do, but because a musical interface needs to be predictable, otherwise it can present surprising behaviours in the performance that ultimately can leave the performer out of control.

Playability test runs are the most important ones, since their observation models most of the behaviour of the environment. It is important to note that a playability test run provides subjective, qualitative results since they are tests done by the researcher alone. However, the subjective improvement of modules should give place to an environment which is objectively divergent, since all the modules that are subjectively tested need to share one same common language and the environment needs to provide a reliable framework for these regardless of their heterogeneous variety. A playability test run will put a special focus on what it becomes possible, and what becomes impossible once a change is introduced. For the context of the virtual implementation, there was little focus on providing an easy learning curve since this environment does not aim at user friendliness, but at enhancement of music as a divergent activity. However, it was considered important that access to changes were fast enough to facilitate a fluid and varied performance.

Throughout the environment development process, a small number of user tests took place. Same as the playability tests, a user test also provides subjective outcomes, and the

value of these tests were the unexpected insights that another subjectivity would provide. These tests were not frequent because the environment being difficult to understand, required an expert subject and a long time of instruction. Since the focus of this development was on divergency, the interesting part of the test would only start after the user acquired a certain familiarity with the environment, and how to use it through a *Calculeitor* controller.

There is a tension between user interface and facilitation of divergence: as seen in the case of the *Korg Electribe*, a friendlier interface encompasses a limitation of possibilities in the same degree. In this exploration, there was a rule that discriminates user interfaces that need to be addressed from the ones that are not priority: the user interface features that are intended to make the interface easier to learn or understand would not be prioritized, unless they were clearly hindering the performance. The user interface features that improved the feedback of the environment's current state and also the features that allow more fluid changes are considered important, however, because these boost the fluidity while improvising music with the interface. Having developed a specification of the environment, products with a friendlier interface can be created afterwards.

One of the most important limitations of current music composition tools that motivated this work was the impossibility to modulate melodies into different scales, chords or transpositions in the real-time. This idea was also explored with the modular *mono-sequencers*. The most important module for on-going composition, is a sequencer. The sequencer would need a clock, and a *preset-kit* module in order to create drum patterns in a fluid manner. Finally, to apply sequences to melodies while answering to the idea of pattern modulation, a *harmonizer* module was created. This module allows transformations on an event-message to respond to musical scales. In playability and live performance tests, this set of modules already provided an interesting environment to improvise music, despite that the modifications to the sequence were slow and difficult to produce.

One of the challenging aspects to decide consisted on the granularity of the message design. In the very first implementation of this environment, each message had an attached output destination. In a single sequence, each message could be destined to any other module. This logic was changed to messages which had no output specified, but are sent consistently to every output of the module. Although the older messages which specified a destination seemed to open some possibilities, it posed more important limitations: in the musical performances that tested this logic, there was the constant issue where re-routing a message path through modules involved several steps. For example, if a sequenced message was being sent to a *preset-kit*, and the intention was to re-route it to a *harmonizer* it would involve the re-sequencing of each step in the sequence to change the route, whereas by using messages of unspecified destination, a single step of re-routing the module is enough. However, each module having distinct registered outputs, could potentially route messages based on the output number, in case a module required. This same distinction will still be present in any potential hardware implementation of the environment.

Another important change regarding the mechanics of the environment, derived into the

creation of the *backward-propagation* concept. This concept derives from the live recording style that is present in most digital sequencers; this consists on the ability to record any pattern which is played gesturally into the device's sequencer by pressing only one button. The modular composition environment, being modular, posed a challenge on how to capture these live performed patterns from any module into any other module. The need to play a pattern by hand in real-time is of evident importance, since this composition method is the most fluid, and ultimately most intuitive because of its similarity to gesture-based instrument designs. The challenge lies in that modules which produced output such as a *preset-kit* or a *harmonizer* are independent from modules that can record patterns such as a *sequencer*. A traditional sequencing tool has the advantage that recording can be a dedicated procedure; but in a modular environment this is not possible because not recording module nor the performance module needs to assume tailored procedures one from another, specially because a recording module needs to serve the function of a performing module in some circumstances, thus allowing to re-purpose the elements.

To create a generic method of casting events from one module to another, a specification was needed. The specification is meant to allow any module to record into any other module, or not cause problems if the receiving module did not have such functionality. For this implementation there were two potential candidates: either the recording module could capture the output of any other module, or the performing module could record its output into any other module. The first approach had two logic problems: first, in order to capture a live performance into a recording module, the user would need to access the recording module, which does not result in a fluid interaction pattern.[14] Second, this method is the same as in the current MIDI sequencers which have record function. By using input recording, it is not possible to discern notes that are only meant to be played from the ones that are meant to be recorded, leading to the impossibility of having a module to process multiple streams of events while recording only one of them. This problem seems specific, but for instance, in groove-boxes such as Maschine or Electribe, it is not possible to have them sonifying MIDI events and record real-time events at the same time, because their recording function also enables recording of their internal MIDI input stream.

The second approach of having the performing module to record into another module also posed a challenge: it implies the addition of another feature to the environment which consist of a parallel network of recording connections among modules. In order to cast a recording from a preset-kit to a sequencer, for example, a recording output could be set from the preset-kit with the sequencer as destination. This idea was tested by using a special function header that indicates that the role of the incoming message is to be recorded instead of performed, and creating functions on each module that would emit event-messages to the modules that are their inputs, henceforth each module becomes capable of recording events into any other module. In case the destination module does not have the capacity to record, this module can either pass the recording message to its inputs, or discard the recording messages.

---

14 This interaction pattern would consist of switching the view into other module and then set it to record, then go back to the original module and play.

A user test run done with an expert in Ableton and live performance using Push inspired the creation of a more fluid and easier to understand interfaces to trigger recording. For instance, the direct access to input recording in the buttons on the bottom were inspired by the experience of this person using the environment; but most importantly the recording protocol acquired its concept of being a 'backward' feature when he was overwhelmed by being able to record to any other module in the environment. This test led to limit the possible recording inputs to only the modules that were actually connected to the module in question (see Fig. 35). This *backward-propagation*, however, remains as a user interface improvement that is specific to the Virtual-Modular environment. In a physical implementation of this environment, however, the recording network could remain as a parallel network that allows the casting of events from any module into any other module.

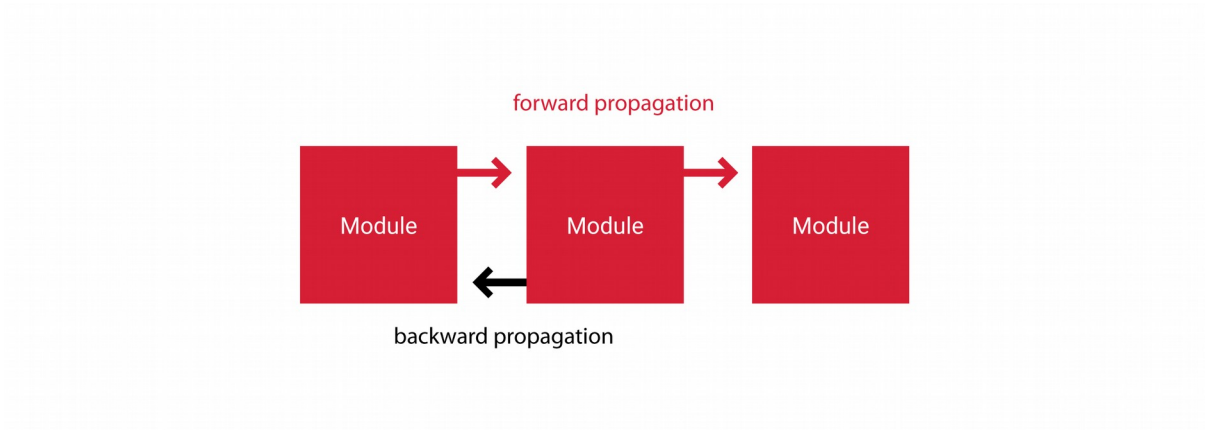


Figure 35: Backward propagation among modules

One interesting thing to note about the directionality of the communications is that, although in this Virtual-Modular implementation each message has one forward component for the obvious communications and one backward component for recording, in a hardware context, these constrains could be omitted, allowing each module to have several inputs and outputs; each for a different purpose, in a way similar to how modular synthesizers do. In a hardware version of this implementation it will be possible to see different inputs, and outputs, each one with a different label, while other more *conventional* modules may offer a connectivity similar to the one of Virtual-Modular.

Some other changes and features in this interface were inspired by already existing performance tools, apart from the ones already described. In the case of Maschine, for example, the arpeggiator feature was a major motivation to implement a performing arpeggiator, which translated into the creation of the narp module. Also the ability to bounce an output into a track for further manipulation incited the idea of creating a bouncer module, which allows the recording of a modified sequence into sequencer, in a fashion similar to live bouncing.

Ableton push has given some hints about how to make a better harmonizer module, and as mentioned before, about how to do live recording. This device offers a scale mode that still

allows the use of the non-scale notes while in the mode, giving the role to the scale of being just a modification of the user interface. “In Chromatic Mode, the pad grid contains all notes. Notes that are in the key are lit, while notes that are not in the key are unlit.” (“Ableton Manual: Using Push” 2018). The amount of buttons also allow for an effective melodic sequencing of events, which is not practical in 16 pads matrix. The disadvantage of this, is that the button sizes are not the best for drum performances. One hint that was applied to the current harmonizer module was the two different views: one in which each semitone uses the space of a square, and other where each square occupies a diatonic grade, having each pad coloured according to its harmonic relation to other notes.

A point that needs attention, regarding to the use of clocks in a bus, is the possible delays in any chain where there is more than one clock-bound module. As it was described in the initial experience with *composite elements*, the response of modules to clock events can become inconsistent according to the order of execution of the signals. As an example, if a sequencer is sending notes to an arpeggiator, the first note to be triggered may happen in the same clock tick if the sequencer is processed before the arpeggiator, whereas it would happen in the next tick in the other case. This usage case is illustrated in Fig. 36 The best solution, as studied there, is to leave the natural behaviour of the delay since other solutions can behave in less predictable ways.

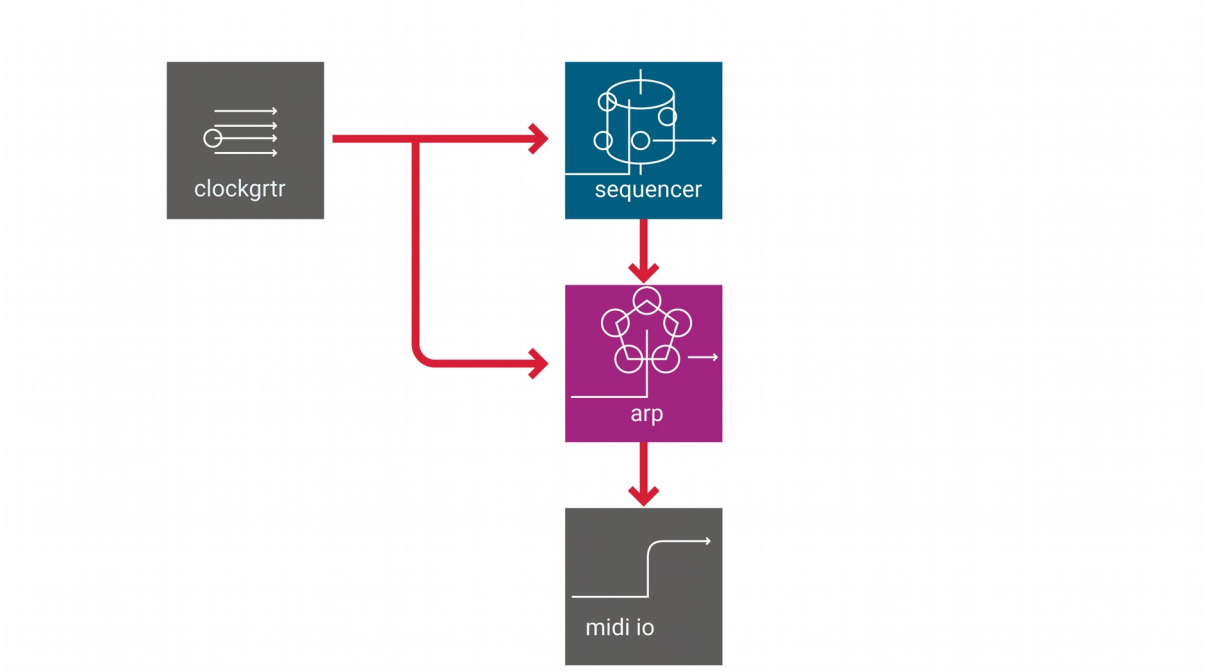


Figure 36: Patch of inconsistent behaviour

Some strategies, however could be applied in the hardware implementations of this environment as to make the behaviour more intuitive. One suggestion is to use chained events, meaning that generated note events could be attached to a certain clock event, allowing a clock-bound module to associate events that are meant to be simultaneous. This chaining could either be done by the use of a header, or making the association if the time

interval between events is less than a threshold time value.

Novation circuit has motivated many future ideas for this environment which to the date have not been programmed into, yet, their implementation is clear and straightforward. For example, a method to clear events from the perspective of the performing module, as a backward-propagation procedure, implied a change in the backward-propagation language by including a header that indicates the role of the event in question: whether it records or it deletes. It also opens the possibility to do other changes through this medium. For a fact, a module can indicate recording state changes, which would allow a sequencer to adjust its length to the recording time, for instance.

As a last remark, the implementation of a composition based integration of micro-timed events such as triplets or swing has been suggested by features found in the sequencers of Elektron, Novation Circuit and Squarp Pyramid. This has motivated the development of modules such as *multi tape* and *piano roll*, which to the date of this text, are unfinished.

4.6 Environment futures

In the development of the Virtual-Modular environment, it was set as a rule that modules could only communicate to the extent of potentially digital messages. This is why it is possible to think of physical implementations for each module that was tested, knowing that it is possible to implement those as stand-alone hardware. In a physical implementation, opposed to its expression through a Calculeitor, could offer dedicated user interfaces that make it easier to understand the roles of each interface element and dedicated controls where it is needed. These dedicated interfaces could function as stand-alone units, or as rack mounted modules, as shown in Figs. 37, 38 in the cases where the function is highly specific. As an additional idea, most of the hardware implementation of these modules could take part in the virtual implementation of the environment as well since it is trivial to include a serial to USB interface in the same way Calculeitor does. It is also interesting that these devices could be used as standard USB-MIDI controllers or stand alone modules in case a user stops being interested in modular composition.

There is an important additional advantage that hardware implementation of the modules will have in comparison to the virtual implementation. Given the nature of the controller in the Virtual-Modular environment and its development history, the modules were thought as single input and single output modules. This helped providing an easier way to control the routing of the modules since this routing was done by using the button-matrix interfaces. One first insight that gave birth to the idea of multiple output and input modules was by the realization that a more complex module, formed of *molecular* elements can have one potential input per molecule that forms it. Additionally, the realization of the recording network, in the context of a physical implementation concretized the need of more than one input per module.

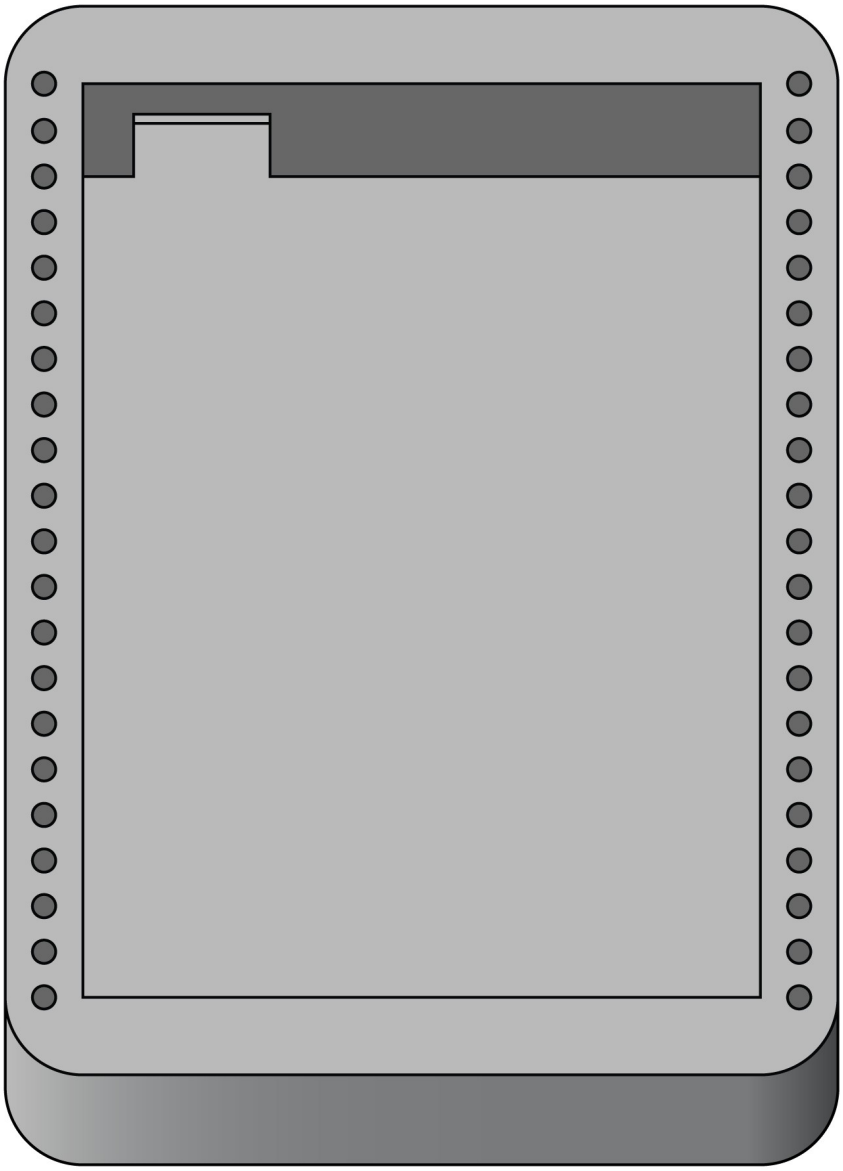


Figure 37: Concept rendering of a composition rack

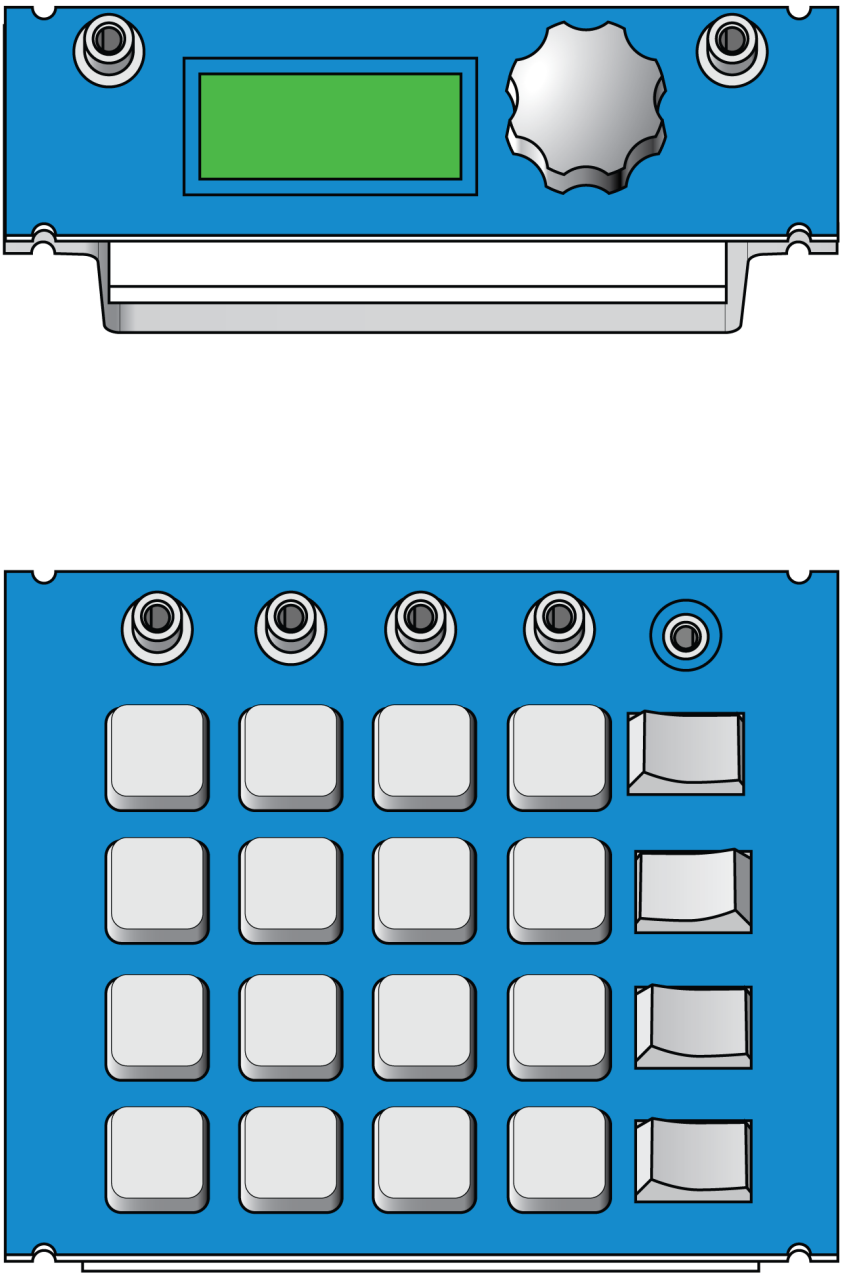


Figure 38: Concept rendering of composition rack modules

The physical version of a module that can most easily fit into the calculeitor interface style is the sequencer. As demonstrated by the Synthstrom's Deluge ("Deluge" 2018), an interface with more matrix buttons affords a friendlier interaction, specially for non-quantized rhythmic features, and the composition of melodies that need to represent note lengths. Such interface could have a similar morphology to the one represented in Fig. 39.

Figure 39: Concept rendering of an extended sequencing module

A module that would improve significantly from a hardware implementation is a harmonizer, mainly because a horizontal distribution of the tonal grades is more intuitive than a matrix distribution. A harmonizer module could have back-lit keys similar to a keyboard, but without black keys, since these are dynamic. Another approach to a harmonizer's interface is an isomorphic, hexagonal keys keyboard. The linear keyboard interface would make of the interface an intersection between a keyboard and a guitar fret, which suggests a form factor which can be placed on top of a desk as well as held like guitar. Additionally, a keyboard interface could afford a mode where each one of the keys being pressed are triggered by using strum controllers, producing a simplified version of a guitar interface. These user interface elements together would appear in an object as displayed in Fig. 40

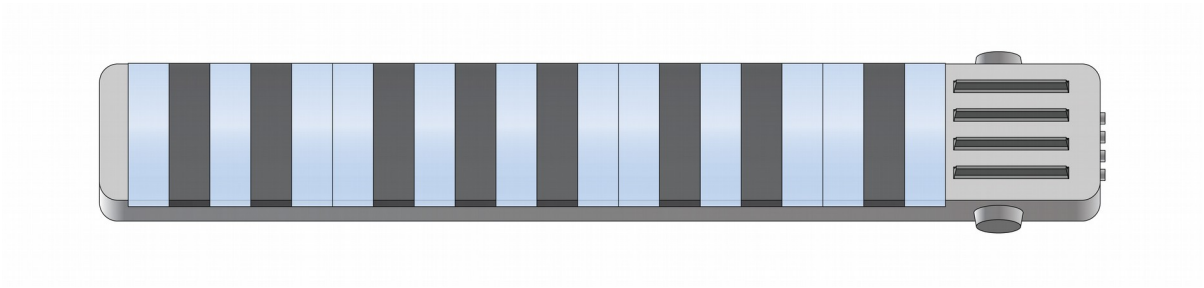


Figure 40: Concept rendering of a dedicated harmonizer and keyboard module

There are usually two stances that people may have in relation to the use of an environment: the dogmatic and the pragmatic, both of which need to take place during the development of a new creative environment. In the ambit of Euro-rack there are many musicians who take a dogmatic stance where they forbid the use of any digital method, following a *dogma of the analogue*. Opposed to this, the pragmatic approach takes each different ambit as an opportunity and allows itself to switch between different working environment however it is more convenient or inspiring. To develop a creation environment that intends to stand on its own, a dogmatic stage is necessary since the self sufficiency is part of the assessment criteria during the development. The earliest concrete musicians, or the earliest synthesists would work exclusively with their newly discovered expressive mediums as if they were the only mean to possibly create music. This allowed the creators to explore the limits and expressive possibilities of their techniques. For a technique to become an integral part of a music system, however, it needs to integrate with other performance paradigms that are seemingly contradictory, but allow them to take part in the greater context. Current electronic music making combines very often the techniques of the concrete music with the ones of the synthesists. Sometimes with the techniques of classical music. The following stages of this project should also involve entering in the area where it takes part with a greater musical composition context, by using different concepts than improvisation, modularity and discrete messages.

Musicians who do not intend to perform with exclusively improvised music could take advantage of the flexibility of the environment to mutate their prepared tracks on stage to greater extents. In this case a musician needs the possibility to prepare patterns that are tied to a set of sounds. This ensures the musician that a certain musical piece can be reproduced on stage, ensuring it will sound the same way it did on their studio. This can currently only be done if the musician has an outstanding memory to recall exactly how to configure the environment with a given synthesizer to reproduce what he intended. This context suggests that the environment would benefit from the ability to store and recall patterns; which is possible by providing any mean of memory recall to the hardware and to the virtual implementation of the environment. This also suggests that certain modules in this environment will benefit of integrating sequencing and synthesis in the same way any drum machine or groove-boxes do, producing a closer relation between sequence and synthesis.

In the Virtual-Modular implementation environment, the preset-kits work as a translation from a simple event-message, to one event-message that has enough information to

become MIDI. In the speculated module that integrates the function of a preset-kit, sampling could already integrated in the interface, as conceptualized in Fig. 41. It is recommended for a synthesizer-provided hardware module to define a set of filter-defined triggers mapped to a set of synthesizer related triggers. For instance, a sampler could define 16 event-messages of consecutive numbers, and have those mapped to 16 different samples; or a lead synthesizer could map consecutive event-message numbers as notes in a similar way to MIDI. Additionally, the synth could define additional triggers to change sound parameters in a way analogue to MIDI control change messages.

Figure 41: Concept rendering of a sampling-preset-kit module

To give a greater scope of use to one device, it would be advantageous to integrate a built-in sequencer so that the device can also run without the aid of any other module. For such device to be integrated in the environment, there is an obvious requirement to have inputs and outputs of event-messages. The question raises on how to integrate the intervention of other modules into a device that already has an attached sequencer. For this, the module needs also to be able to ignore its own sequencer input, and instead route this sequencer into an event-message output, so that a module can be *side-chained* between the sequencer and the sound module. Therefore, an integrated sound and sequencing unit must also permit the same functionality than the two units could present if they were separate.

Deejaying devices carry a big vernacular load, hence a modular environment approach to deejaying can only provide with means to produce a similar effect and workflow as the

one of deejaying, but not offer a meaningful improvement as a device for deejay culture. Presentation of pre-recorded tracks brings the value to more popular audiences of providing recognizable songs or patterns, which for the broader audiences is crucial. The modular environment could add to live-remixing, the benefit of complex patterns of beat slicing, and jumping around a song that goes beyond a mere loop lock. For instance, a track sampling device can offer the possibility of completely re-arranging a track according to an emerging sequence, that allows modifying an on-going, recognizable song into a new one that only shares the timbral characteristics with the first. This possibility could lead to a hybrid between live composition and Deejaying.

Figure 42: Concept rendering of a loop-oriented sound module in the spirit of the modular composition environment

Being both serial based protocols, it is easy to understand that the modular environment protocol can be translated into MIDI. The event-message function header is translated into the first nibble of the MIDI header byte; and all the rest of the transformations are only recommendations: to use the event-message's third number as the MIDI channel, and the second as the MIDI second byte (note or CC parameter). This changing of places is graphically represented in Fig. 43. The event-message's fourth byte is recommended as the MIDI third byte, which accounts for velocity or CC value. This is because event-messages are not required to carry velocity. The need to express a three-bytes midi message in four bytes in the modular environment accounts for the need of these message to be purpose-agnostic, meaning that a message whose functional parts are separated in bytes are easier

to re-purpose in a modular patch than MIDI messages, which use the header byte for two purposes. For more complex modules such as sequencers or sound modules, midi input and outputs could be integrated in the unit. A dedicated conversion module could have an interface similar to the one rendered in Fig. 44.

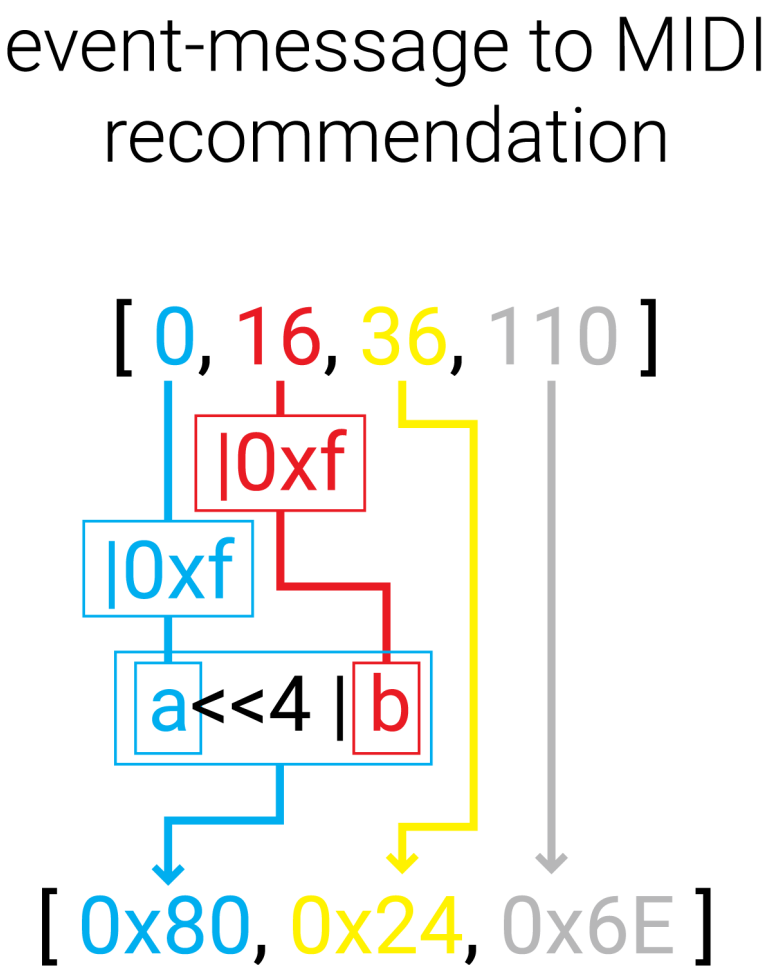


Figure 43: Schematic representation of the conversion from modular event-messages to MIDI messages

Another obvious translation from the environment's event-messages would be to control voltages, to facilitate integration with modular synthesizers. This is the main reason why for the modular connectors of this environment is recommended to use a different connectors than 3.5 millimetre jacks. This reduces the risk of confusion between two signal types (digital and analogue). An event-message to analogue converter could require additional mapping settings, since the requirements for patching may vary, and a good design approach would be to implement three different mappings that can be switched with a single knob or switch, as appears in Fig. 44. One example of such mapping could be

a MIDI-like note on and off scheme, where the second number selects the destination plug, and the third or fourth bytes define the voltage. Another example could be mapping the third number (which mapped to MIDI would translate into channel) to select the physical cable, while the second number defines the voltage level.

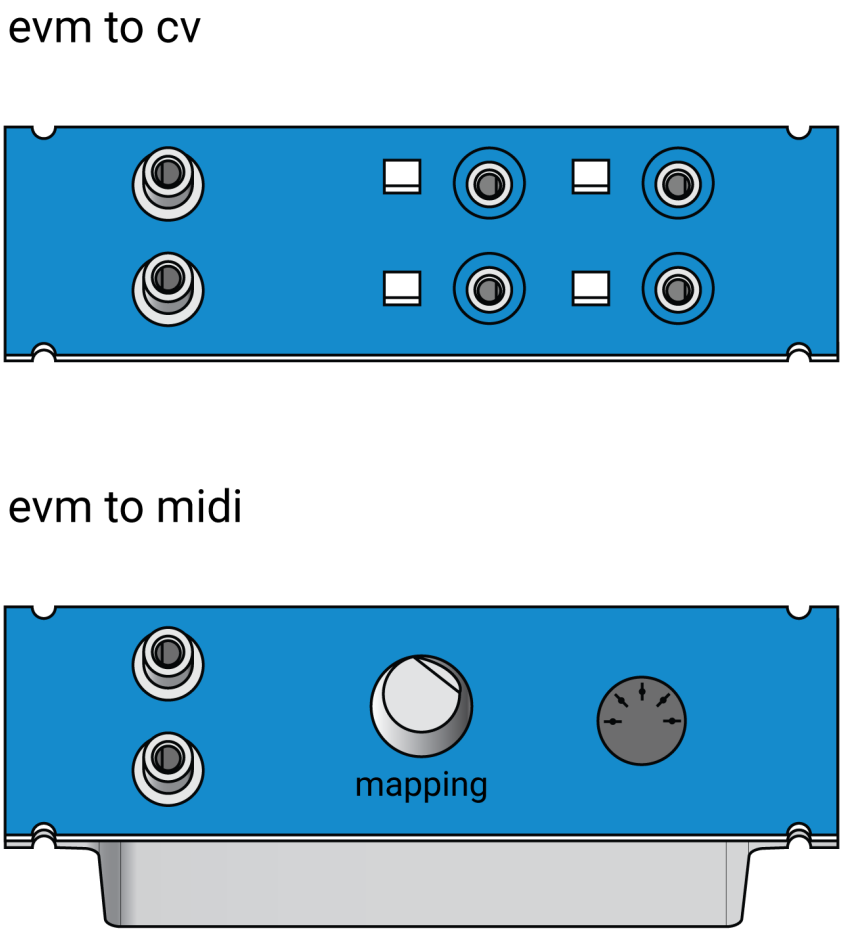


Figure 44: Two concept rendering of rack conversion modules

The Korg Kaoss Pad 2 served as an inspiration to consider a module which produces control signals. In one hand, serial based protocols are prone to overflow, as it will often happen to a MIDI stream when a detailed control messages are sent. However, such a device could produce lower serial signal rates to reduce the control signals rate, while having fully analogue output voltages sending control voltages at a higher sample rate. This can provide an interface between a quantized environment with the other continuous value environments. In this way, an infrequent digital signal, can trigger another high-rate continuous signal, thus producing a bandwidth efficient approach to translate parameter change signals into analogue signals. This module could also integrate its own effects

processor, just like the one of the Kaoss Pad, but with the ability to externally drive the effect parameters, and with more focus on looping these automations.

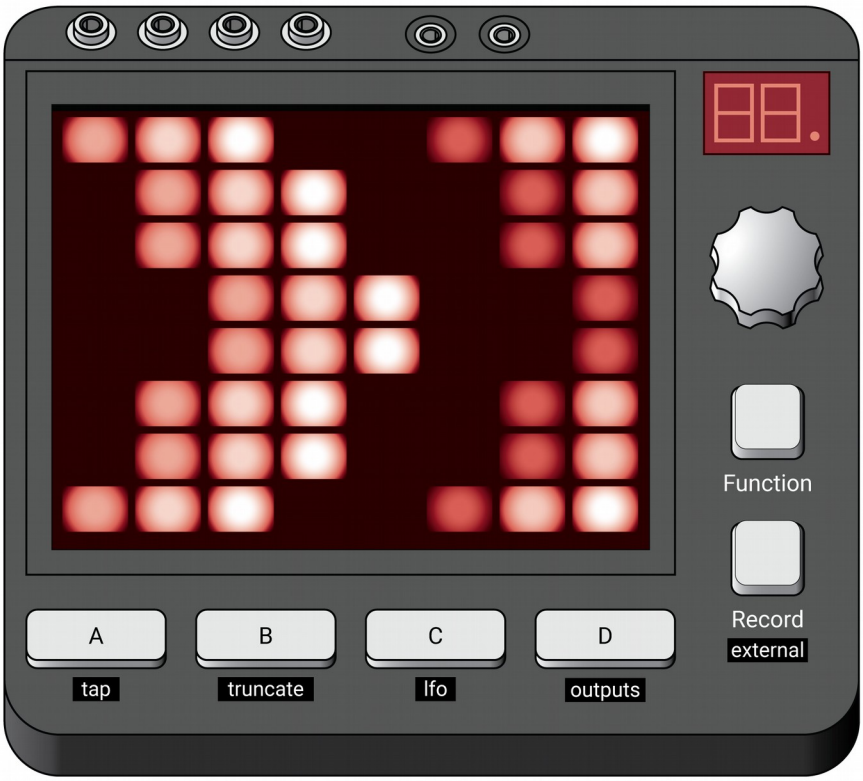
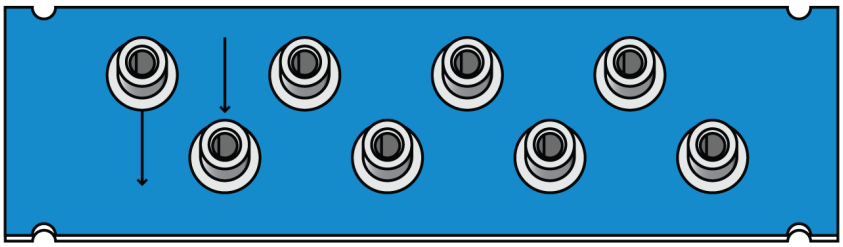


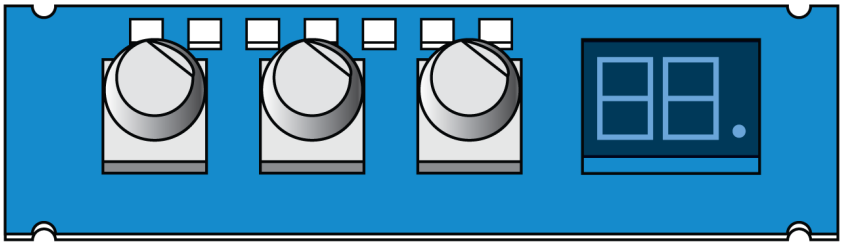
Figure 45: Concept rendering of a module that makes reference to Korg's Kaoss Pad

A signal merger and splitter can be simple device. A serial signal can be sent to many devices as long as none of them draw the voltage down. For this, the splitter could ensure the levels of the signal by using discrete components. The more complex operation of a merger/splitter device is to merge two incoming signals. In this case, all the incoming signals need to be stored in a buffer, and sent to the output one after another, starting by the oldest. The merger/splitter contains two rows of plugs; the first row being inputs, all get merged into a single stream which is cast directly to all the connectors of the second row. The resulting concept of module would look similar to the representation present in Fig. 46. If the user needs only to split a signal in two, a simpler device could be used where the cables are merely connected without active components. This would allow a signal to go through two different paths forward in the patch.

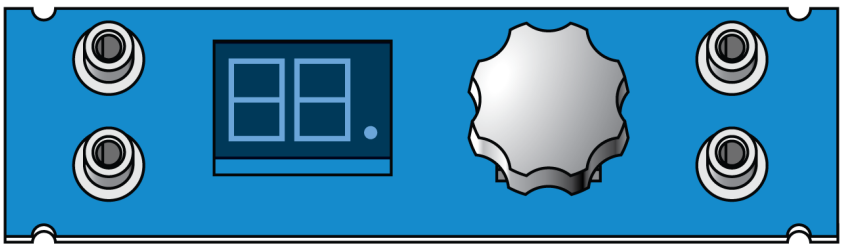
merge



operator



arpeggiator



clock generator / clock sync

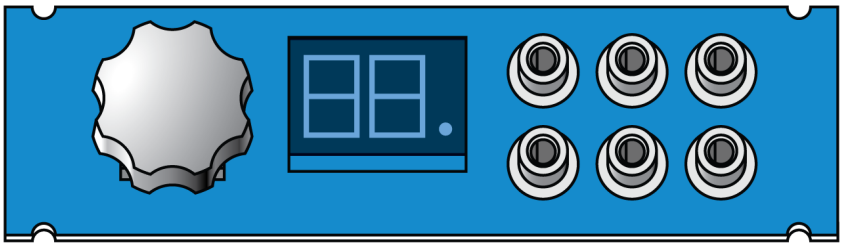


Figure 46: Renderings of devices with highly specific functionalities

5 Evaluation & discussion

This chapter contains three different assessments of the success of the modular environment in question. The first section evaluates the effectiveness of the modular environment in real parties where it was used. The second section describes some examples of different musical systems that can be built with the environment, as a measure to assess the effective divergency in the layer of musical systems. Finally, a comparative assessment is done in order to describe the divergency of the modular environment in comparison to other techniques of performing live conventional electronic music.

5.1 Experiences performing with Virtual-Modular

During the development process of the *Virtual-Modular environment*, there were many opportunities to test the concept and its current state of development in its intended field: a dance party. These are the best opportunities to test the highest level effects of the *composition environment:* the music in a social context. The performances helped drawing conclusions of its effectiveness at different development stages.

5.1.1 Fukuoka-shi, Japan

For this performance the Virtual-Modular environment was at the earliest state where it could be used to perform live. In this performance it was realized that the environment's interaction patterns needed to add a focus on fluidity, by offering default behaviours, since the time it took to configure each event and pattern was long, and the musical outcome came out very repetitive. In other words, it is not enough that an interface allows to do a certain operation, it is also necessary for such operations to execute fast. Thankfully, there was a drum track being sequenced in a Korg Electribe, which reduced the strain on the composition interface. For the most part, the melodic content was being generated in the interface by heavily relying in emerging polyrhythmic features, while a conventional drum pattern was being generated by the Electribe synthesizer.

5.1.2 Ääniaalto, Helsinki, Finland

Ääniaalto is a yearly festival of sound art and performance, which is a perfect opportunity to show projects that propose something new such as this one. At the time of the application to the event, the Virtual-Modular environment was very advanced in the version 2, having performed previously in Kyushu with a version 1 of this modular composition environment.

The visual feedback that was available in the performance given at Kyushu was no longer available because it was designed to work with an older version of the environment's prototype. After a small demonstration of the prototype in thesis seminar course, it became clear that the Calculeitor's interface did not give any clue about the aspect of modularity present in the tool, hence it was decided to adapt the visual representation of the environment to work with the newer version, so that people could understand and visualize the musical operations.

The visuals used in Ääniaalto were based on the ones prepared already in Kyushu, but this time, an additional layer of information was added for better clarity: a layer that draws a different representation for each module type that would take instance in the modular environment. The presence and connections of the modules were the same as previously, using a D3 (Bostock 2017) force-directed graph, but a layer of a Konva canvas drawing plug-in ("Documentation | Konva - JavaScript 2d Canvas Library" 2018) was superimposed, allowing more easy addition of texts and graphics that are unique to each module. A protocol of communication between the Virtual-Modular environment had to be devised, so that the graphical interface could account for each module, its type and its connections to other modules. The graphic interface also represented the messages the modules would exchange, and the lengths in the case of the arpeggiators and sequencers.

During the performance, the control of the environment was lost, causing the performance to end prematurely, due to an unexpected factor. The problems that caused this failure, were more related to psychological factors than to technical issues. Despite that this environment has been used without problems in laboratory conditions; with the pressure of an audience, it became difficult to find a correct strategy to escape from an error. The initial intention was to produce a scheme of muting the drum pattern to do a melodic change and then bring the drums back, producing a spontaneous change. The preset-kits were set to mute with this objective in mind. To produce a fast melodic change, an arpeggiator was created at which point the sequencer that was responsible of the drums, was deleted unintendedly. After re-creating the sequencer as fast as possible, the transcurring time increased the mental pressure. When the drums were unmuted, there would be no resulting sounds, perhaps because of a missing connection or a wrong setting in the synthesizer. At that point, the pressure was such that the performer decided to give up. In many performances with audience, this performer has faced problems such as computer shutdown or software failure; in all these cases it was possible to exit the problem state easily by disconnecting the affected device and using another device instead, as backup. What is interesting of this *emotional failure*, is its inescapability: despite that any technical difficulty could have been addressed simply by using the backup synthesizers, when the emotional state of a performer fails, there is nothing they could use as backup of their own mind.

The burden of the interface in the failure hovers over this experience: the insufficient information in the interface was clearly one of the factors that caused this failure, but also the lack of confidence was important. For instance, with a much simpler composition system such as Maschine, it can take more than a year of practice to attain the level of confidence where it is possible to figure out alternative solutions when there is a noticeable problem in the sound. To get familiar with the more complex modular environment, at least the same amount of time would be expected. An additional factor that might have affected the mistakes in the performance, are the many changes that have been applied to the user interface, which caused the mode of operation to be constantly changing.

This problem, thus highlights the need of focused practice, but also highlights the importance of finishing the development of the hardware version of this interface. In the current Virtual-Modular environment, a module can be hidden when it is not focused in

any of the controller hardwares. As a consequence, events can happen without showing visible evidence in the user interface. In contrast, in the hardware implementation of the modular environment, there would be a one-to-one relation between hardwares and module instances: each physical unit not only represents, but is one module; meaning that less information can be hidden from the user.

5.1.3 Calculeitor party

The 4th May a party was organized exclusively to test the Virtual-Modular environment. Another intention of the party was to provide additional social networking opportunities as to improve the chances to build product out of this thesis project. The party took place in Aalto's Kallio Stage in Helsinki, and two other participants were invited to play dance music to offer a more varied set of music and set the desired framing to the party. After the realization of this event, a short feedback interview form was handed to the participants, and the written responses were collected.

Feedback form Questions:

- Your name (or anonymous)
- In what ways the music performance was changing, and in what ways was it constant? How does it compare to other electronic music performances you have seen?
- Why did you decide to come?
- What were your expectations, how did they compare to the actual event?
- What can you tell about how the musical instrument is used to make music? How did you know/realize that?
- Any other comments

The evaluated success of the party is ambivalent. Despite the most important aspects of the party took place, namely, the presence of audience and that the music during most part of the performance was improvised using the Virtual-Modular environment. One of the aspects that were not achieved well, is that participants did not dance beyond subtle body gestures. One of the participants of the audience, Camilo, who had knowledge about organizing events commented about a lack of bass in the sound (Sánchez Carranco 2018). Another aspect that affected the framing of the event was the presence of chairs. It was presumed that some other important aspects were hindered by characteristics of the space where the party took place. In one of the feedback responses, it was mentioned that it was hard to know what to do during the party because of the contradicting presence of dance music, and chairs in the space.

Despite these drawbacks, the party allowed the audience to get fully engaged with the ongoing music performance, and as a result it was still possible to assess the interaction between the created music and the audience. Some questions of the feedback form revealed some interesting perceptions from the participants. For instance, it came clear that the music performed is perceived as different from the usual, although still being

conventional. It was mentioned by a participant that the music had both: largely repeated patterns and surprising changes as well. These observations account for the fluidity and originality of the performance.

The Virtual-Modular environment could offer an approach to recover the performance relatedness with the audience in electronic music. A common situation in electronic music performances is the unawareness from the audience about how the music is being made. This happens because, opposed to mechanical instruments, electronic music instruments have non-obvious relations between interaction and sound. This is considered an issue because it reduces the interaction between performer and audience; as pointed out by one feedback response where the person thought that there were prepared preset patterns. By the question "what could you deduce how the instrument is used to make music" it became clear that there is no clarity on how the music is made but the development of visuals is a possible development path that could strengthen this relatedness. This idea became apparent from mentions in the feedback form as well as in conversations about the visuals. There was an active involvement trying to infer the relation between the visuals and the music, and if they represented the state of the environment in greater detail, it would provide the audience with an intuitive understanding of the inner world of the performance.

5.1.4 Kaiku Pheromondo

After the Calculeitor party, an invitation was received to play at a party named *Pheromondo*, in Kaiku. Being two weeks after the Calculeitor party and with the feedback of some of the assistance in the audience, it was possible to improve the response of the music to the social group.

The Pheromondo granted better chances than the Calculeitor party to assess the potential of the modular environment. There were the advantages of a longer time, and a party environment. Additionally, there were two hours to play, which provided enough time for the performer to calm down and observe the audience's response to the musical changes. To a greater extent, it was possible to model musical changes according to the observed reactions of the audience, and replicate the successful modulations at different points during the performance. Not having access to request the audience to fill a survey, the success of this performance was only reflected by the constant dancing, the involved response from the audience to the musical changes, and the positive impressions verbally manifested during and after the performance by people that were not aware of this project.

5.2 Systems exploration

As a way to evaluate the flexibility and originality possibilities that the Virtual-Modular environment affords, this section describes some of the meaningful musical systems that can be built in the environment, and used in a performance.

5.2.1 Introducing a drum kit

One sequence can be used to play any set of sounds. If a sequence was originally intended to play a certain drum kit, its output can be routed to a different drum kit. This adds some variety to the patterns without requiring to program new patterns.

To switch into a set of sounds without making a disruptive change in the sound, the output of the sequencer can be switched to a preset-kit that has all its sounds on mute. After the switch, the preset-kit can be taken off its mute, sound by sound. Depending on what synthesizer is being used to produce the sounds, it is also possible to introduce the new sound by fading-in the new sounds gradually. The same can be done for melodies. By interposing a preset-kit between a sequencer that contains a melody and its output, it is possible to mute all the grades of the melody, and gradually unmute them.

5.2.2 Polymeter

The modular environment can easily form polymetries, since the sequence lengths need not to be interlocked. An easy method is to program a single note on a narp and change the playback rate while it is running. This produces an interesting rhythm that jumps from polymeter to normal, or to off-beat patterns. It can also be done by using arpeggiators, when the amount of arpeggiated notes are not multiple of the other sequences. Additionally, two sequencers with different lengths can produce emergent polymeters. This technique is broadly used in electronic music either by using a polymeter of 3 against 4, or by using a beat synchronized delay with a delay time of 3 beats. It was observed that polymeters of 5 or 7 against 4 are also interesting and easy to listen.

As an addition to the described polimetric system, it is possible to attach an additional sequencer which re-starts one of the two sequencers, making these to come in synch every certain number of steps. This technique makes the musical patterns easier to understand rhythmically. To produce this feature, an additional sequencer can be added which, connected to the out-of-meter sequencer, sends jump signals with an interval that matches the main metric. To exemplify, let us think of a composition based on 4/4 meter, with a second sequencer which runs a sequence of length 3. In this case, both sequencers will produce a cycle of length 12 (the sequence repeats every 12 steps). In case of techno or house music, a length 12 is not highly expected. Many of these tracks produce a *forced reset*[15] of this polyrhytm at step intervals which are multiples of 8 (most commonly 16 or 32). To achieve this reset, a sequencer is added and connected to the secondary sequencer, and a trigger on is programmed to trigger every 16 steps, with a number 0. This trigger on event, effectively causes the secondary sequencer to jump into step 0 every 16 steps, thus producing the desired polyrhythm reset.

---

15 Meaning that the secondary sequence step is set to 0 regardless of its current position.

5.2.3 Held note

Many MIDI synthesizers have what is called a *MIDI panic* command. The phenomenon related to this feature can be used in the environment purposefully. The *MIDI panic* command shuts off all the notes that were left sounding indefinitely. These notes are called *hanging notes*. Using digital signals to represent note events imply that either the system only uses note on events (implying that the duration of a sound cannot be expressed), that all the notes have an inherent duration, or to use note on and note-off events. Because it is the simplest approach, the latter is being used by MIDI, and was adopted in the Virtual-Modular environment as well. One example of using a hanging note purposefully, is to leave a *hanging note* on an arpeggiator, as exemplified in Fig. 47. It can be achieved by first adding a source (e.g., a sequencer) of note on events to the arpeggiator (1), and disconnecting that source after the note on, and before it emits the note-off (2). This causes the arpeggiator to lock in an arpeggiated pattern, until its memory is cleared by the user. For *hanging note* security, the MIDI output module keeps a list of the notes on, and it is possible to send all the matching MIDI notes-off from the list.

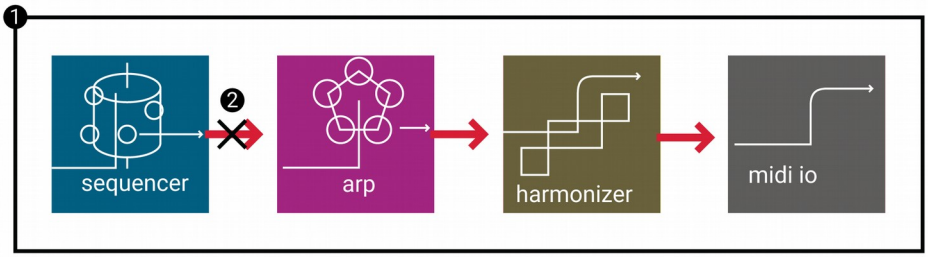


Figure 47: How to produce a held note in the Virtual-Modular environment

5.2.4 Skip-jump sequencer

The sequencers of the Virtual-Modular environment are designed to jump to different parts of the sequence when they receive a trigger on event. This makes it meaningful to connect one sequencer to another, as illustrated in Fig. 48 Alternatively, the sequencers can be set to stop when they receive a trigger off. This allows for interesting patterns where one sequencer can cause other sequencer to jump into different sections of a pattern. This technique is similar to what is possible in the sample-based technique called beat-slicing, except that in this case it is applied to sequences. Two sequencers can trigger each other in a loop, to produce unexpected generative patterns.

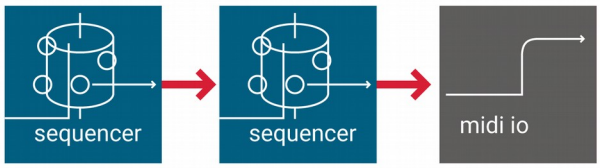


Figure 48: how to produce a skip-jump-sequencer system in the modular environment

5.2.5 Patternized arpeggiator

By setting the clock source of an arpeggiator to something different than a steady clock it is possible to do more interesting dynamic patterns. The virtual version of the modular environment, by default creates a bus which outputs to every other module. This is specifically intended to have a clock being distributed by default to every module. To create the patternized arpeggiator, first an arpeggiator is created, from which the main bus is disconnected. This causes it to stop running. After disconnecting the arpeggiator, an additional sequencer is added, having its steps programmed with clock events instead of notes. The resulting chain of modules is illustrated in Fig. 49. Now the arpeggiator advances one step every time the sequencer triggers a clock event. This technique replicates the type of arpeggiators that have a pattern options, such as the pseudo-arpeggiator of the *Electribe 2*.

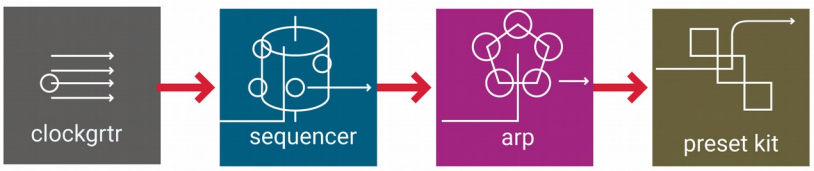


Figure 49: How to produce a patternized arpeggiator system in the modular environment

5.2.6 Toggling note

Sometimes it is needed that certain notes in a sequence vary from one repetition to another while having the rest of the sequence running consistently. This can be set up by creating a secondary sequencer or arpeggiator that contains all the variations of that event. Each of these alternating notes can be triggered by the main sequencer, when it sends a clock step. For this, the main sequencer needs to be connected to the secondary sequencer through an operator which lets only clock signals to pass. In a way, this pattern is similar to the patternized arpeggiator, with the difference that in this case, the same sequencer is used to program triggers and clock events. This system is illustrated in Fig. 50. This system could also be potentially used to create Elektron style conditional triggers ("Analog Four Manual" 2018, 36).

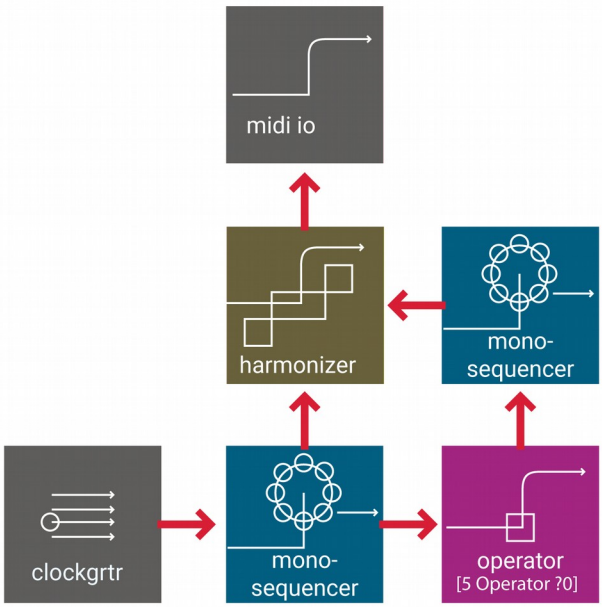


Figure 50: How to produce a toggling note system in the modular environment

5.2.7 Progressive melody

A progressive melody is similar to the toggling note in the sense of having two sequences producing a longer melody by using operators as an interface. In this case, the main sequencer is used to play a constant melody, and a secondary sequencer that is running at a portion of the first sequencer's rate, causes the main sequence to transpose.

It is possible to apply this technique to an already running sequence, without interrupting the melody. The indication of the Fig. 51 suggests that one normal sequence could be playing through a harmonizer (1). To this scheme an operator and an additional sequencer are added in parallel, but not connected to the output (2). The operator is set to operate the note number, for instance, with addition (3). The value of the operator can be such that it does not produce a change (e.g. +0). The main sequencer route is changed to pass through the operator (4). The secondary, slow sequencer is connected to cast events into the operator by using a bouncer (5). The operator becomes a note modulator which is constantly changing, according to the programmed pattern in the secondary sequencer (6). At this point a dynamic transposition is applied to the main pattern. In the illustrated example, note that depending on whether the transpose operations are caused after or before the harmonizer, the transposition can be chromatic or diatonic, respectively. Such type of patterns are common in blues, for example, where a melody is repeated four times but in different transpositions of the pentatonic scale.

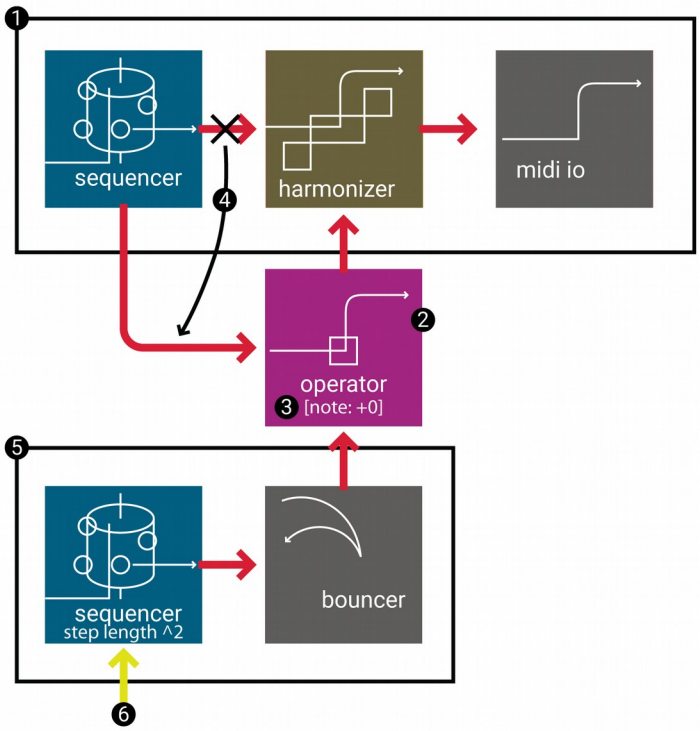


Figure 51: How to produce a progressive melody system in the modular environment

After producing the progressive melody construct, often connecting the slow sequencer to the fast sequencer will produce interesting results, because the slow sequencer will cause unexpected jumps in the fast sequencer, causing an emergent new melody on the base of the first. This same pattern can also be applied to the output of arpeggiators and other similar modules, perhaps to generate complex harmonies whose elements are all modulated by the same rules.

5.2.8 Sequenced pattern routings

The route-sequencer allows for many unusual composition alterations. One of these is the possibility to apply transpositions or feedback delays to a musical pattern variably depending on whether it is on its strong or weak time. By the same means, it is also possible to subtract or silence all events according to this rhythmic role simply by disabling the route on the desired steps.

As a function of feedback for a delay module, it is possible to produce generative patterns by feeding back some of the route-sequencer's output modules back to the delay. These modules between the route sequencer could be such as, for example, an operator or a chord generator, thus changing the composition of the pattern stream. A note-sustain module can be interposed between the delay and the route sequencer as an effective mean to limit the amount of events (preventing an excessive amount of events).

A more usual application of this module, is to produce swing on any composition. As it was specified that different effect routes could take place depending on the rhythmic role of the event, events in a weak step could be routed to a delay, thus producing an effect of swing. It is also possible to produce less usual rhythmic artifacts, such as a swing where only one per each four steps fall in the correct time.

5.2.9 Feedback loop

Alike other modular environments, it is possible to produce feedback loops. This dictated the use of lazy queues instead of a call stack when it came to the communication between modules. A lazy queue consists on a list of tasks that need to be performed, which are processed in the same order as the tasks were queued. While in the context of an event stack, the causation of a feedback loop leads to a stack overflow error.[16] In a lazy queue, however any amount of events can be added, with the effect that a feedback loop may cause an ever-growing queue of events to process.

The effect of producing a feedback loop of modules which are clock bound, generates unexpected musical patterns, sometimes with very long periodicity. Feedback loops comprised of non clock bound modules produce an explosion of rapid events which create harsh noise and glitches in whichever module is interpreting the control signals (e.g., Pure-Data receiving MIDI). This difference is caused due to that clock-bound modules expect clock signals to send outputs, while the not bound modules propagate the signals as fast as possible.

16 When a function calls another function, it is referred to as being *stacked*. This is because the caller function is expecting the called function to return (end) in order to proceed. If this stack gets too large, it causes a stack overflow error. The stack limit is usually very high, hence stack overflows are usually caused by procedurally stacked functions, specially in functions that call themselves, such as the case being described.

Adding a feedback loop to a delay module can lead to interesting results. Note, however that in this case is crucial to add some operation that could remove events after a certain amount of repetitions; otherwise the events accumulate fast and slow down the performance of the environment. In this sense, it works the same way than any delay module: if the feedback does not attenuate the signal, the noise accumulates until distortion. A good way to do this, is to use two operators in series between the feedback output's and input: one that subtracts from a number, and other that lets pass only events whose number is larger than a certain amount. It is also possible to use other modules such as a route sequencer, which would propagate the events only at certain intervals. In the chain that modifies the delay's feedback, any module can be interposed, which can lead to different alterations to a note which vary upon each repetition.

In addition to the performance patterns that were just described, many additional ones could be built. Moreover, it is still possible to create new modules that could open new possibilities in this respect. It is speculated that either the physical or graphical implementation of this modular environment would allow the creation of more complex musical systems: the connections between modules, currently being selected and visualized through the button matrix, are difficult to understand. Higher-levels of complexity that would be trivial by means of visible modules and cables.[17] All this accounts for the broad divergency that this tool offers: it is possible to generate many different musical systems, each of which affords a spectrum of musical results. This provides two dimensions of musical expression in the live stage: creation of systems and creation of musical patterns. In addition, a third dimension of divergency is added when considering that the author can code his own modules in the context of preparation.

5.3 Comparative assessment

The initial question of this thesis was stated in the terms of how a composition environment could afford more divergent improvisations of conventional electronic music. In relation to this divergency, three metrics were described: fluidity, flexibility and originality. As it was mentioned, these metrics were important in terms of how the tool affords the expression of these three characteristics. According to these, it is possible to assess the success of the created environment in comparison to current music improvisation tools. To assess the value of the new composition environment, a comparison can be made against other current live composition tools of different natures.

5.3.1 Fluidity

The scope of fluidity on a live performance, more than having relation with the amount of sounds produced, it has relation with the amount of musical ideas. A live performance is

17 Visible modules and cables can be achieved using a graphical user interface for the VirtualModular environment, or by creating a physical version of the environment.

most likely loop-based, and hence a new musical idea is represented by changes to that loop where the repetition of the same loop is considered as a permanence.

Referring back to the performances based on gestural mapping, it was seen that improvisation is possible within parameters of body coordination and agreement across musicians. This thesis project included the development of a physical interface which offers means to produce music from commonly used gestures. Modules such as the harmonizer or preset-kit offer the common press-sound relation between action and sound. The environment also offers more complex results to the same simple gestural operations such as transforming tonality, composition and rhythm in different ways. However limited the possible gestural inputs in the Calculeitor controller that was devised, it is not difficult to imagine the creation of devices that could capture more complex gestures. A good guide to such development could be Imogen Heap's working prototype, which suggests the use of position, posture, touch and gestures among many other gestural variables (Heap 2013). In this sense, a very interesting new research question opens. It would be about the exploration of composition procedures that gather body and gestural variables, and attain musical meaning within the context of modular composition. A symbiosis could be attained between the live composition of musical systems and live performance using these musical systems.

With regard to a deejay performance, from the point of view of an unaware audience, many musical changes are present, since these are integrated in the recording. This can convey a sense of fluidity. From the point of view of the deejay, however, it is necessary to think in a different abstraction than composition or scoring, since there are very constrained possibilities to re-compose a pre-recorded track. Focusing on potential,[18] there is only a limited amount of not performed variations to a pre-recorded track such as applying a filter or jumping to a different point of the song. There are also constrains on how to superimpose pre-recorded tracks because of these are an already rich musical piece: both tracks need to sound well together. In this sense, deejaying has an ambit of improvisation which corresponds to the choice of layers, and application of effects. It excludes the ambit of composition improvisation.

Performances with DAW based tools such as Maschine allow very fast input of a composition since it can be played with precise pressure sensitive pads. The possibilities to modify those sequences once performed, however, are limited, unless they are modified using the mouse and screen interface in the laptop. In the case of Ableton; there are more modulation options and there are more interaction patterns available to play, such as chord playing, chromatic, more arpeggio patterns, more effects, and so on. Moreover, it is possible to apply transformations to the composition stream using Max as a MIDI effect ("Creating MIDI Effects" 2018; "MIDI Effect Tools" 2018). However, the max patch itself cannot be modified using the physical interface but max abstractions could be chained as midi effects. If each instantiation of MAX in the Ableton context is counted as a module of this project's environment, it could be said that MAX within live, using Ableton push, is a

18 As it was explained in the introduction, potential can be described as how many other musical compositions are *not performed* once a musical composition has been chosen.

predecessor of this work. on the other hand, if each module in a MAX instance accounts for one module of this thesis environment; the use of Calculeitor largely improves performative fluidity. In both cases, however, the physical implementation of the composition environment that was speculated would improve the possibilities for a user to produce musical contents fluently thanks to the presence of physical connections that relate the different modules.

In relation to current live programming tools, programming generally takes time, resulting on compositions which progress gradually. For composition of conventional music, however, the best approach would be to construct instruments that are performed on the live, and combine the command-line programming interface with a dedicated controller to produce faster, more abrupt changes. The concept of command line plus dedicated controller could lead to interesting results. The Virtual-Modular environment in combination with Calculeitor controller could be developed to become (and in a looser sense it is) a programming environment plus a controller performance tool. In its state of modular environment, however, still allows a fluid modification of the patch and in performances it proved being able to generate many variations. Since the interaction is expressed in multiple interaction nodes, more than one change can be produced at the same time, whereas in programming, only one thing can be written at each given time (it is possible, however, to postpone many changes to one single moment, or to associate many outcomes to one single variable). The possible implementation of additional interfaces for the modular environment e.g., with better playing pads, or mapping of other gestures could enhance fluidity while keeping the current characteristics.

In terms of fluidity, consequently, the product of this project offers a unique potential in terms of composition, because it affords to easily produce or alter musical ideas during the performance.

5.3.2 Flexibility

Within one performance, deejaying performances can present many variations but these tracks will come with their immutable characteristics, as with any other track. To give more flexibility in performances, Native instruments *Traktor* introduced the concept of *stems*. It consists on the commercialization of musical recordings in separate tracks, thus allowing deejays to manipulate the pieces further. This comes along with their own software support for such type of tracks. This practice could be considered similar to the use of Ableton clips in live performances. These pre-recorded material, however, still possess the constraints of a sample. In the scope of a single performance it is possible to use enough variety of recorded material, so that nothing is repeated throughout the performance. For an audience, there would be an appreciation of musical flexibility (all the presented loops in the performance are different) but the artist may be aware of the authoring possibilities on their set. Additionally, parts of two performances of one same artist may end being almost exactly the same, as it happens in shows such as The prodigy, Daft Punk or Stephan Bodzin. In these cases, the repetition across performances is intentional as to account for their own tracks.

With modular synthesizer environments, theoretically anything can be done, but, as explained earlier, some conventional composition features are not trivial to achieve. Conventional music making in Euro-rack is often characterized by this fact. Live coding environments such as Sonic-pi sonic pi, being environments too, allow a very wide range of musical outcomes, to the extent that is possible to make experimental music that transgress the conventional composition abstractions. This variety can only be achieved by adding custom programming abstractions, since building more complex patterns via textual commands takes time. The Virtual-Modular environment could be viewed in this sense as a set of prepared abstractions which can be combined in different ways within an environment, and modified through a physical interface.

While performing with DAW paradigm tools, it happens that despite the patterns may be different, the alterations often end up being always the same. One example of this, the difficulty to change the patterns on an Electribe. In this case, the performances with Electribe are almost always limited to switching among prepared patterns and alteration of timbral characteristics of the sounds in the pattern. The same happens with the more advanced interfaces such as Push or Maschine. The repetition of modulations is less noticeable in these cases because there are more available modulations and with a greater scope of pattern possibilities. This may grant two applications of one same procedure to two different patterns to sound like different alterations. One illustration of this is that in Maschine it is possible to transpose any pattern one octave up or down. Transposing a drum pattern results on the same pattern on a set of sounds that are different but related to the original (playback rate was doubled). The application of this same technique to two different drum kits may not be noticed as the same modulation by a listener.

In the case of the current Virtual-Modular environment, it seems that there still are boundaries with respect to the possible modulations. Despite the modularity, live performance may still be limited by a three-dimensions boundary comprised of the available modules, the available procedures or parameters on each module, and the time it takes to set-up an intended composition system. In this way the limitations are in practice similar to the ones of DAW based environments, specially Push (since it is the most complete DAW) but theoretically offering an additional dimension which broadens the boundaries of improvisational divergency. In practice this reveals that future works with the environment are paramount to enrich the composition possibilities: user interfaces that allow more fluid and clear interconnection among modules will expand the possibilities in the *composition systems* layer.[19]

In the current state of the environment, it is not possible to store and recall musical compositions. This feature is theoretically possible and could provide an additional source of variation. By recalling patterns or fragments of patterns, it would be possible to produce drastic composition changes, similar to how it is possible by playing a new track on a deejay deck.

In conclusion, the modular environment improves the potential for flexibility within its own specific area. While prepared performances can produce more abrupt musical

19 refer to Fig. 7.

changes by using playback, the modular environment allows *some* abrupt changes by the transformation of current musical material, however, without the need of any prepared material. In this sense, therefore, the modular environment provides a wider playground for flexible musical composition, since these transformations can be chosen on the live performance instead of them being determined beforehand in a preparation process. In addition to all this, the speculated futures of this environment could provide with the possibility of producing prepared abrupt changes by the combination of the environment with sampling techniques.

5.3.3 Originality

The Virtual-Modular environment, in its current state, affords the performance of an original composition, because in addition to the composition freedom present in performance paradigms such as looping, it offers an additional dimension which is composition of a musical system. It can be argued that the Virtual-Modular environment offers a set of procedures in the same way than any DAW based composition paradigm. As it was discussed before in this work, however, it is possible for musicians to prepare their own modules or tweak the behaviour of existing modules to produce their own systems. In the case of the speculated physical implementation of the environment, it will be possible to make use of lower-level composition modules which can account for a less constrained range of possible musical systems. In this way musicians can attain their own signature musical modulations or composition systems, and improvise new ones during a performance as well.

Originality in deejaying has more than one aspect. In terms of the live music production, it is very likely for a well informed audience to recognize tracks across different performances. Against this, there is the vast variety of tracks constantly being composed and published, which deejays could resource to. Some artists compose their own tracks. As it was discussed at the beginning, many deejays sought originality in their deejay performances by modifying the recordings to further extents, by removing the labels from records, or by looking to the most obscure producers to pick up on their sets. This leads the deejay performance to become original in a different degree to the performer than to the audience. While a participant of the audience may perceive that she has never listened to the tracks being played, the deejay is aware that these come prepared beforehand. Originality in deejaying can also be attained by using unique techniques and features. Carl Cox, for example has his signature shout “oh yes, oh yes” (Cox and Beyer 2018). In addition, he is very active in the tweaking the music flow by fast and intermittent fades of volume, and cueing of tracks (Cox and Beyer 2018). Originality, therefore, can be attained at deejaying in terms of original deejaying techniques, but hardly in terms of composition. The product of this thesis, in its current state however, cannot make use of musical tracks in the same way as it is done in deejaying. In this sense, there is a whole area of live production which is still unattainable, yet it is possible to imagine concepts where a prepared track paradigm can be integrated with the modular paradigm, as it was discussed before.

Other tools offer different extents of originality to a live performer. For instance, a tool with very limited composition possibilities impose to a greater degree an identity of the machine to the piece. One example of this are the Teenage Engineering *Pocket Operators*. Some other performance systems offer the authors a vast creative area. Examples of this are Ableton or live coding environments like Tidal or Super Collider. The performer can resource to prepared music-altering procedures. In case of these being created by performers themselves, the prepared modulation procedures could grant an aspect of musical identity to the modulation algorithms with which their performances are provided.

For more limited tools such as Maschine, the aspect of originality is again different experience for the audience than for the performer. In these tools, alike almost all the other music making tools, it is possible to have prepared sequences which, being exclusively created by the performer, will appear as original to the audience. These performances may, however, be similar one to another of the same performer in case he resources to the same prepared patterns, across more than one different live performance. In the case of the Virtual-Modular environment, it is possible to prepare some musical systems beforehand, which in turn is a composition environment that possess a field of possible musical outcomes. The initial system, however, can be altered in the real time, allowing to drift out from possibly known composition procedures.

The modular environment, therefore, offers the same affordance for originality in terms of conventional music composition in the live stage than DAW based or loop based tools. With this environment, however it is possible to create completely new musical transformations which are not possible with the other mentioned tools, because it is possible to improvise the modulation systems in a way that is not possible by using other tools. Furthermore, it is speculated that this affordance can be enriched by integrating new modules, creating new user interfaces, or by expanding the versatility of current modules.

6 Conclusion

Having created and evaluated the modular music composition environment, the relevance of this project in relation to the field is evaluated. In addition, the conclusion reflects upon the different processes of this work and their effects on its result. This evaluation also leads to some conclusions which reach beyond the scope of this project, and hopefully can be insightful for future design processes.

The modular environment, thus is not a good replacement of most of the current tools for performance, because each performance technique has different objectives in mind, which may not necessarily be divergency. The environment rather has the potential to bring a way of understanding live performance of electronic music. For example, divergence is appreciated by the audience on how the live-ness of the performance is sensed, otherwise a sampled performance already presents a greater divergence. The environment, thus is an improvement only where more creative freedom is intended in the ambit of composition.

The fluidity comparison with Ableton considering the possible use of MAX brings an interesting point of view to the idea of developing the environment physically: parallel to how in Max and Pure-Data there are control or digital signals versus DSP signals, the composition environment could be considered like the missing digital side to modular composition in order to turn modular synthesis into a more complete environment, as MAX and Pure-Data are, with the added benefit of a much better and multi-point interface. In this sense yet another new research path is opened, this is, exploring the integration between continuous and discrete abstractions of modular music composition. This exploration would be possible both, by using software, or using hardware modular synthesizers.

The composition environment could enhance the sense of live and authorship. Be it using the virtual environment provided that a clear representation is displayed by showing the on-going operations (in a way analogous to the visible code in live programming) or by using the physical implementation of the environment. The presence of a controller whose feedback is shared between the performer and the audience shifts the ritual of a laptop performance into a ritual of instrumental performance. Thinking of the classical performances with mechanical instruments, the awareness of performance from the audience is given by the visibility of relation between the performer's actions and the musical results. In the same way, a visible manipulation of digital composition devices could lead to an increased sense of live performance in comparison to other live performance tools.

According to the comparison with other performance paradigms and tools, the composition environment seems to still offer boundaries to what is musically possible. In its current state of development, the environment offers improvisation possibilities perhaps comparable to using combinations of other tools. This is caused by the currently used interface, which makes the construction of patches non intuitive and limited. An improved interface would account better for the relation between modules, and specially, provide a plurality of input and output connectors. In such case, the versatility of the environment would provide with a greater extent of composition possibilities, above any current digital tool.

The initial insight that gave a starting point for this thesis was understood intuitively, and there was some work to be done in order to understand the exact description of the problem through different processes. The initial inspiration was driven by a frustration

while using currently available live performance tools. Being aware of a subjective frustration despite that these tools are technologically advanced and well thought was intriguing and also problematic as how to define what a *better* tool should be. The initial process of surveying all the other tools available in the market and reviewing their manuals finally provided with the needed insight, as described in the *Musical devices and their performance paradigms* chapter. As said, these tools only offer musical modulations when there are dedicated procedures to them. This briefly defined fact was one of the many surprising revelation that emerged from this project. This process of surveying also served to give a scope of development, as it was realized that tools divergent music making already existed, only that they were not oriented to conventional music making. This further underlines the utility of surveying current alternative approaches to a certain design problem, be there candidates for the exact same problem, or to a problem which is similar.

The development steps to follow were fuzzy: after having defined a project, there was only the idea of making a digital (discrete) modular environment for music composition. In hindsight, the process that was described here as *Composite elements environments* appears as an arbitrary starting point. The idea of designing a modular environment has many possible approaches, and at that point there was no knowledge about available design approaches for this task. This arbitrary exploration start point had the advantage of providing a view into the intricacies of this design task without yet knowing the best approach. Thanks to this process, a design approach emerged which finally allowed the design process to take course from a less arbitrary starting point, as it was described in the *finding the primary elements of the environment* process, that was successful. It was crucial, for instance, that the exploration with the Virtual-Modular environment considered discrete modules which would communicate using an array of numbers. It is interesting to note however, that the definitions derived from this process were over-specified; meaning that later in the process it was discovered that some different interpretations of the rules could be acceptable (like for example having more than one input or more than one output for complex modules).

It can be considered that from the perspective of designing environments, the composite elements process was a learning process, and the following processes were the application of that lesson. For future environment design projects, provided that the initial information is sufficient, it will be possible to start with the non-arbitrary approach of *buildification*. This *buildification* approach may not be limited to design of physical processes, but it may as well prove useful for other designs such as social or economic processes. The downside of this approach is that it requires knowledge at least of some of the possible systems that are desired from the environment in question. The discovery of this buildification process, was on its own a very valuable lesson to be applied in the future, whenever a design process is related to systems theory.

The aforementioned process revealed that a step previous to the definition of design specifications or methods could be a good addition to the design process of complex products (such as an environment). This additional step would consist of making a *mock-up* project only to understand the complexities of the task in hand, as well as to reveal the designer's own intuitions in relation to the project. In more general terms, it was learnt

that the production of a short project can be part of the process of understanding the problem. A production process, consequently, may not be exclusive to the production phase of a project.

The design of the physical device had some clear limitations which could have only been solved with more resources and time. As it was described in reference to the design of *Calculeitor* hardware, an almost arbitrary decision was taken to use a led-buttons matrix in a similar way to similar devices such as *Novation Circuit* or *Novation Launchpad*. It would have been as well possible to think about embodied interfaces, or a live coding interface, probably leading to different environment concepts. As a project that looks for new approaches to perform music, it would have been interesting to explore the relation between a modular environment and the physical user interaction beyond a controller. In this aspect, the only user interface propositions were taken from the other projects that were surveyed at the start. On one hand this simplification allowed a more dedicated exploration in the *environment* aspect of the product, since such hardware could be manufactured out of standard parts. On the other hand a very valuable aspect of the live presence that these interfaces could produce had to be left for future projects: a gestural mapping interface that heightens the perceivable live-ness to a further extent, as expressed in the *Sound Gloves* research, considering the audience not as mere listeners, but also as another user of the musical interface (Lai and Tahiroğlu 2012).

Regarding the design of the physical product, it is clear that this product does not communicate clearly enough about its capabilities to an unaware user and would not be a self-explanatory product. The design of the physical product had to be limited to what could be helpful for the development process of the environment. This is a clear consequence of having started the process of designing the hardware before defining the design approach. As it was demonstrated by the audience of Calculeitor party, it is not possible to understand the composition elements of the performance without the aid of an additional graphic representation. In this sense, the design of the hardware was more than anything the design of a development tool that allowed the virtual environment to be used in the real context. It also helped determining what is realistic to expect hardware as parts of a network. The exploration about the networks design process had an important impact about what the limitations on the virtual environment experimentation needed to be. Without a clear knowledge of the hardware limitations, it would have been difficult not to prototype environments virtually, which later would be impossible to build as hardware.

The exploratory process with the Virtual-Modular composition environment was one of the core components of the project, where all the details about how modular performance could work, were tested. The fact of being involved in the design of a user interface such as Calculeitor naturally led to the intent of doing user testing. Some tests were conducted with users, but this method further proved that this project should not target making easier user interfaces, but is about the development of a new method to perform live. The user testing became unfruitful because it took too much time for the participants to learn how to use the interface before being able to start with user interface testing. This is because the Virtual-Modular environment is very different from other composition tools, and it takes time to get acquainted with the idea of discrete and modular composition. In addition to that, there was the added difficulty that the patching of modules was not

visible at all times, and this required a highly developed acquaintance with the environment. While it was possible to solve these issues by implementing a graphical user interface that would reflect the environment more intuitively, the focus was limited instead only to the divergent possibilities of the environment. It is clear that users with enough interest can also learn to use difficult instruments, but in order to generate this interest, the instrument needs to offer unique expressive possibilities in the first place. A useful aspect from the user testing, however, were the additional observations and comments. As the tests were targeted to electronic music tool users, each one had their own different views that provided the exploration with a rich set of ideas to develop. These insights now form part of this project in many different theoretical and practical aspects of this project.

One interesting topic that might become of use to other environments such as live programming, is the casting of notes between different notation systems in a way parallel to casting of numbers in programming languages. As expressed in the design of the harmonizer,[20] there is not only one approach to cast a chromatic note into a scale. These different approaches could lead to ideas such as decimal points in a note (a chromatic note numbered 1 counting from 0 can result into a C.05, expressing a C# in a major scale). Such expression suggests the idea of generating a set of musical data-types such as frequency, tone, chromatic and diatonic major, and offering different ways to cast across types. This is not an idea that might be of use in the case of the composition environment, but it definitively is useful in the context of live programming environments.

One of the most useful prototype testing methods during the process were the live performances. As described, live performances revealed the most important aspects of such development by setting it under the intended environment: an expert user and a live dance social gathering. While developing in lab conditions, it is easy to think of performance patters that are highly complex and nuanced because these afford more interesting possibilities. In context, the limits become clear about how complicated the use of the environment can be before causing problems to an expert user. The success metrics also are very clear: whether people remain engaged with the music or not.

Another valuable lessons learned from the live performances is the major role that emotional factors play. The first and most clear example was the failure to rescue a stuck performance in Ääniaalto, where despite it was technically possible to proceed, it became emotionally impossible. Another, less clear example that occurred in performances which were not listed in this thesis but were still based on the use of the Virtual-Modular environment[21] revealed the strong relationship between audience and performer in the case of improvised performances. In two cases where the performance was initiated with

---

20 Where a number can be *expanded* to a scale, providing one input number to each diatonic note, opposed to *rounding* the number to the nearest diatonic note, hence having possibly more than one input number on each output note. See the harmonizer description in the appendix.

21 two underground electronic music parties, two performances in parties related to university activities, and a demonstrative performance at Espoo Mini-Makerfaire.

no audience, it was realized that improvising music for nobody was surprisingly difficult. In these same performances, when an audience gathered to listen and dance, the flow of improvisation became easier and more effective. This last effect was not mentioned in the discussion because beside demonstrating the profound impact of emotions in the performance, it was not possible to discern whether this effect took place as consequence of a real audience to performer relationship or a particular personality trait of the performer. Additionally, the same effect was observed when using other performance tools.

Together with the live performance testing, the environment tests without audience were very predominant to the conception of new modules. Without the pressure of an audience, it was possible to explore into more complicated or experimental patterns which might not necessarily result in conventionally musical results. This testing method provided with the creation process of many different modules, when amidst a performance, new modules were imagined that would be useful in such context. All the modules apart from the initial ones (midi IO, harmonizer sequencer, and preset-kit) were initially thought from these experiences. One limitation of this method is the coupling of the user interface to the characterization of modules. One example of this is how, all the modules are thought as single input and single output modules due to the way these can be patched through the *Calculeitor* interface. Were these modules physical units, or would have these modules been patched via a graphical user interface, the conceived new modules would have had different characteristics, and the emerging composition techniques would be different. The described drawback from this method needs to be taken into account if this environment is translated into a hardware implementation: the modules need not to be replicated in the same way, since hardware interfaces will have different affordance than the virtual one, which was taken into account in the environment future section.

Testing without audience also served to improve the interfaces to achieve better fluidity: one clear example being the *outside scroll* interaction pattern, where it is possible to change a parameter of a module by selecting it and scrolling the encoder, without *entering* into the module. It is clear that the user interface improvements from these tests are limited to the Virtual-Modular implementation. The same *outside interactor* example illustrates this: in a hardware implementation, all the parameters will be already physically present.

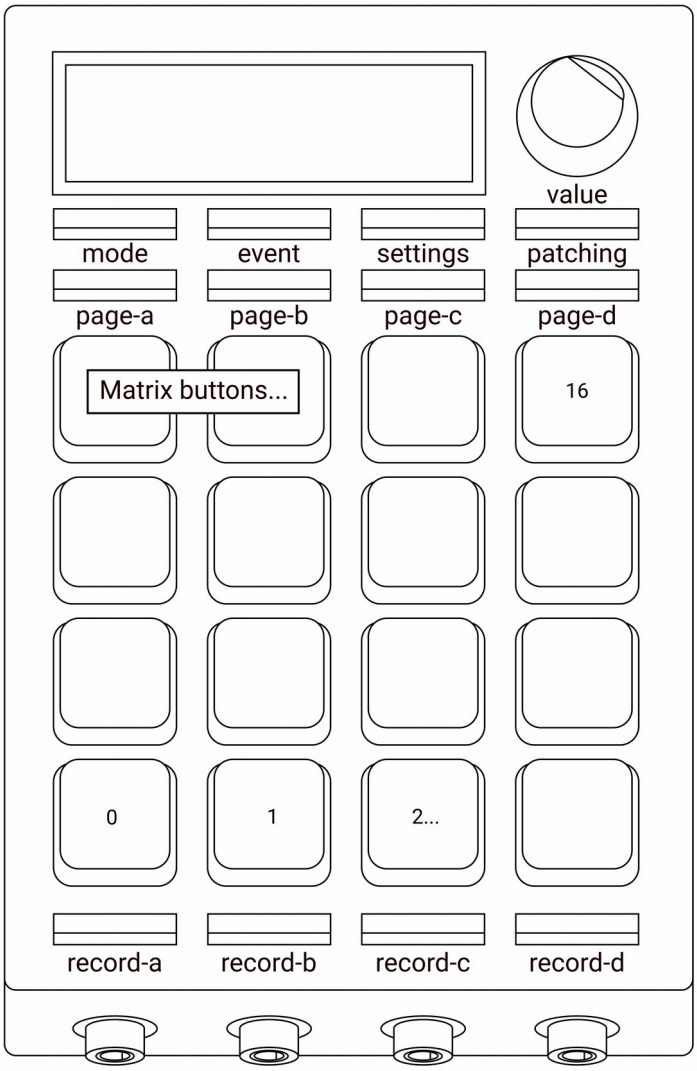
The development of this project led to the production of an interesting new tool to improvise conventional electronic music in ways which may have not been possible before. The more interesting result, however from this project has been the discovery of all these new processes and complexities that are related to improvisation and live performance such as social interactions, the role of emotions, and the extent to which the performer's interaction with the music is noticeable. Additionally, the futures for the environment that was designed in this project transcend the area of interest of this project in particular and could lead to many new areas of exploration such as collaborative composition, modular gestural mapping, and incursions in the mixture of digital and analogue modular processes to the creation of music.

7 Appendix

Additional documents that can illustrate better the implementation of the Virtual-Modular environment, including a basic usage tutorial, and a description of some of the modules that were created. Note that in the case of the tutorials, the name *Polimod* was devised to refer more easily to the modular environment.

7.1 Usage tutorial: Calculeitor interface introduction

7.1.1 Button Names



Function name of each button in calculeitor, which is effective in both, the physical and virtual contexts

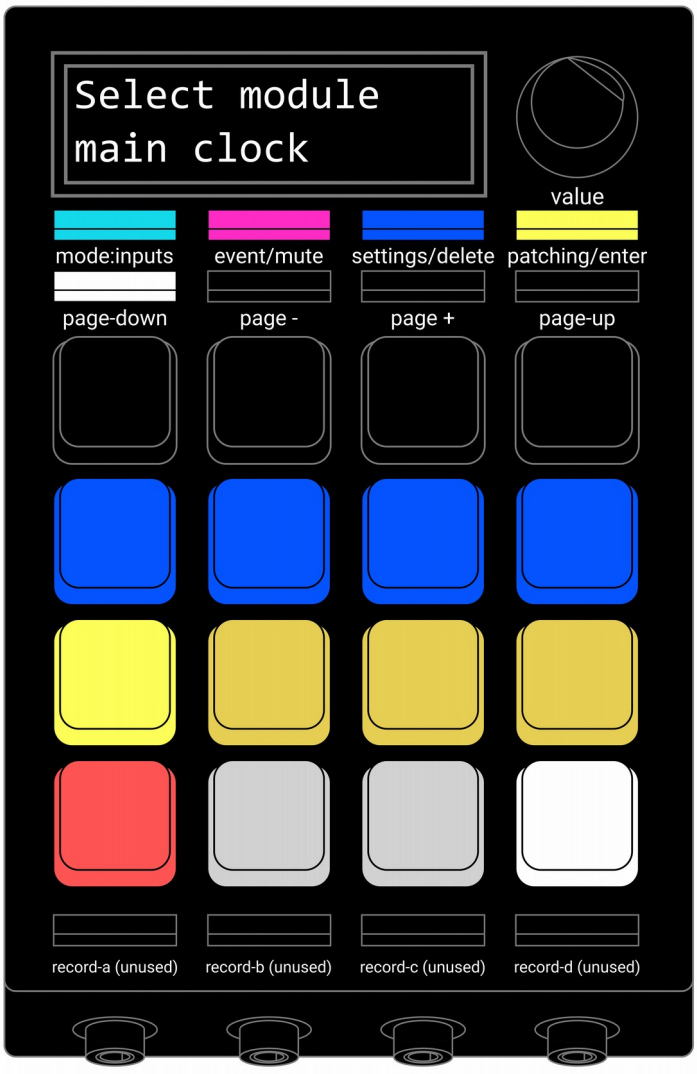
7.1.2 General button functions in a module

This is a general description of the interface buttons for an overview. If you don't

understand something, don't worry, it will come clear later. This guide could serve you as a reminder while you advance in the tutorials.

- Value (rotation encoder): changes the parameter that is being displayed on screen. Rotating it right (clockwise), raises the value of the parameter, and reduces the value when being rotated to the opposite direction.
- Mode: works like a shift button. This will make sense later, but in general terms it changes the function of the buttons, or momentarily alters the module's response when pressing buttons.
- Event: when you are in a module, this button lets you select what message it outputs. When in *super-interactor* mode, this button is used to mute.
- Settings: when in a module, this button displays a settings menu in the buttons matrix. By pressing different buttons in the matrix, different global parameters of the module will be displayed in the screen, allowing to change them (using the encoder). When in super-interactor, it is used to delete modules.
- Patching: this button is used to change between the *super-interactor* and modules. Think of it as the *esc* and *enter* keys of your computer.
- page a-d: if a module contains multiple pages, (e.g. a sequence that doesn't fit in the 16 buttons) these buttons serve to select among them. A bit like scrolling with the mouse, or changing tabs in a web-browser.
- Button matrix 0-16: these buttons are used to perform. Depending on the module and whether there is a menu open, they serve different purposes. The matrix is where the magic happens.
- Record a-d: among the other modules that are connected to the current module, these buttons enable recording. This allows for example, to record a drum loop into a sequencer without having to switch into the sequencer.

7.1.3 Super-interactor



super-interactor

The super-interactor is the main interface of the virtual Polimod environment. In this mode, it is possible to create, open, move and remove modules. The button functions are different in this mode than when in a module.

When the application opens, it will be in the super-interactor mode. Different buttons in the matrix may appear lit in different colours, while some other buttons may appear unlit. These coloured buttons represent one module each. Modules can be selected by tapping them.

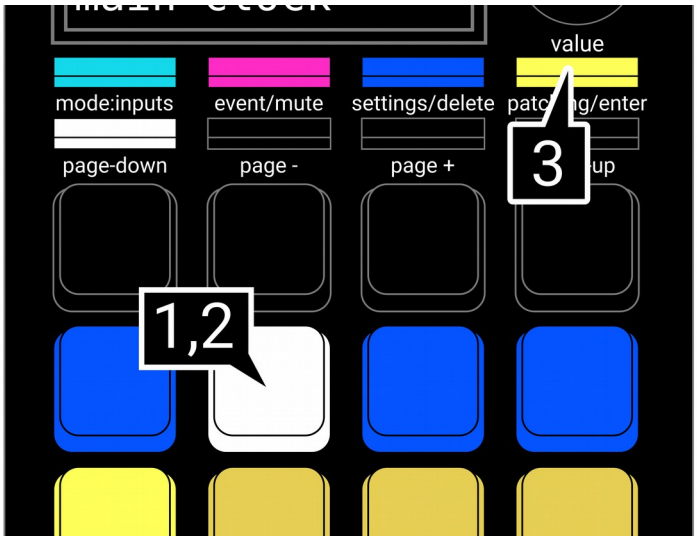
7.1.3.1 entering and leaving a module

When in the super-interactor mode, it is possible to open the interface of a module by

tapping the button that represents the module, and then pressing the “patching” button.

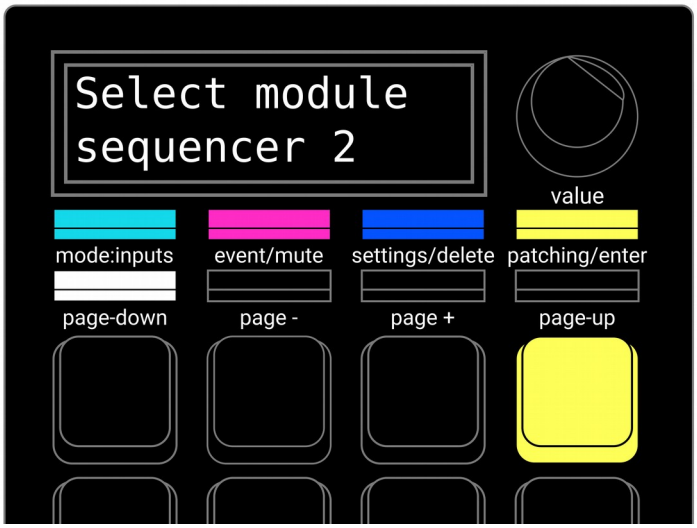
When a module is in focus, pressing the patching button, closes the module and goes into the super-interactor mode

> tip: it is possible to switch from one module to another fast by holding the patching button, and releasing it after the desired module’s button was pressed.



Connecting and disconnecting modules, steps 1-3

1. In super-interactor mode, tap the matrix button which appears blue. That colour usually represents a sequencer.
2. The button matrix that was previously blue turns white
3. Now press the “patching” button.
4. The contents of the screen change, and the buttons matrix turn off, displaying a yellow play-head which advances in the matrix. This means that the calculeitor is focusing a sequencer.

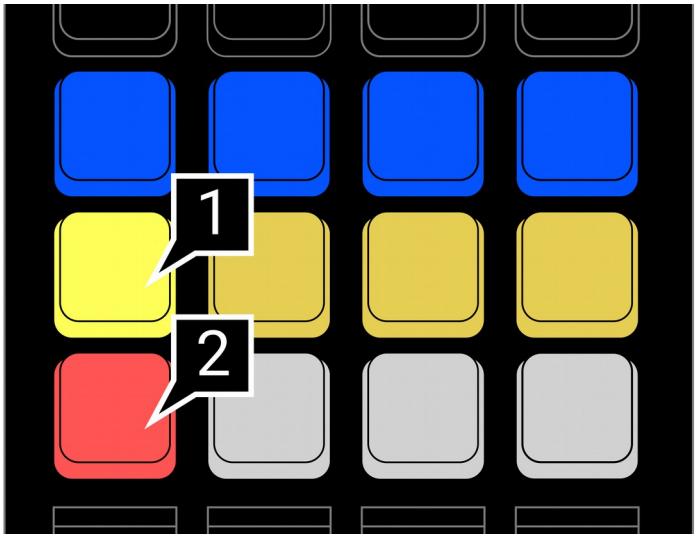


Connecting and disconnecting modules, step 4

7.1.3.2 connecting and disconnecting modules

A relation can be established between modules by using the super-interactor mode. It is only possible to select outputs for each module. Connections among modules can be seen by selecting the module: the other modules that turn red are the modules that are connected as outputs of the selected module. These connections can also be toggled by holding the module's button and pressing the output module's button while that button is being held.

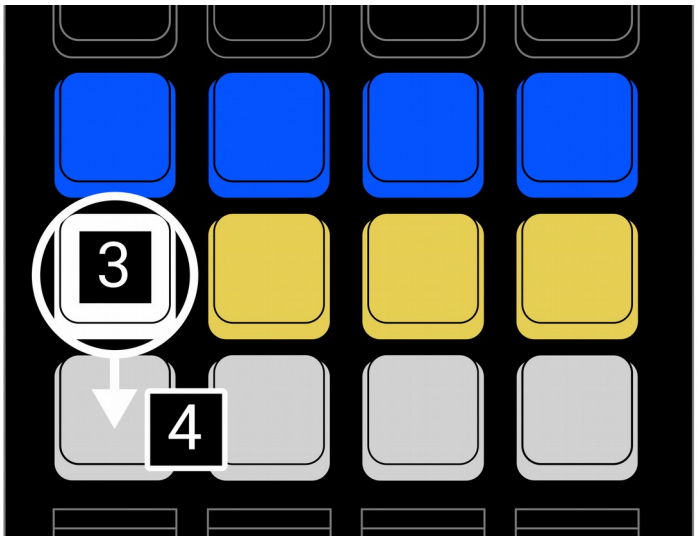
1. In super-interactor mode, select the module named harmonizer. It is coloured yellow.
2. Observe that one module turns red. That module is the midi output module.



Connecting and disconnecting modules, steps 1-2

3. Press the selected harmonizer button again. While that button is held, press the midi output module button (which is currently red).

4. The midi output module button turns grey. This means that these two modules are not connected any more.



Connecting, steps 3-4

5. Repeat the step 3. The midi output button turns red again, meaning that the harmonizer is connected again to the midi output.

6. Repeat this operation with other modules. Try connecting and disconnecting.

> Tip: it is possible to see the input modules by selecting the module in question, and then pressing the mode button. The input modules in this case will turn cyan (light blue)

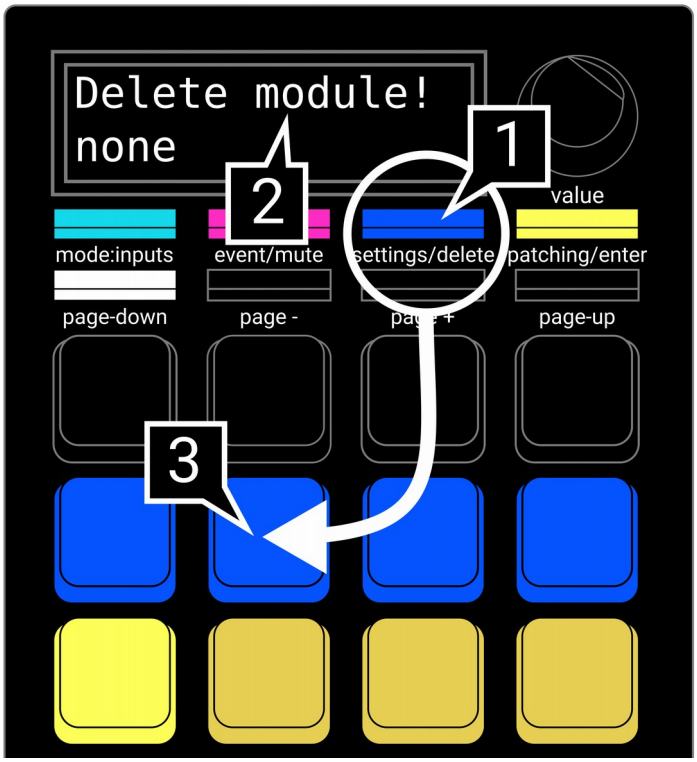
7.1.3.3 deleting modules

Modules can be deleted and created. To delete a module, tap the module's button while holding the "settings" button.

1. In super-interactor mode, press the "settings" button, which appears blue.

2. The screen displays the text "delete module!"

3. Press one or more than one module button in the button matrix, while still holding the "settings" button.



Deletion of modules, steps 1-3

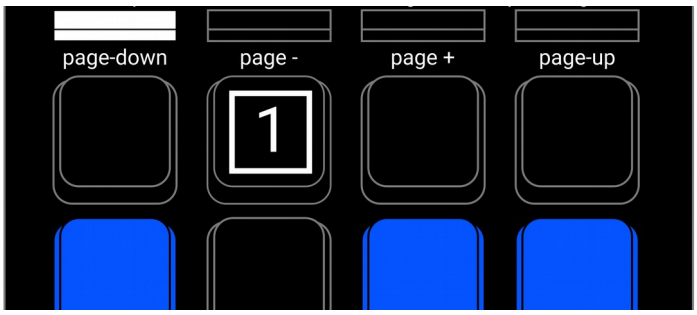
4. When the “settings” button is released, the selected modules will be deleted.

   Tip: when a module is selected for deletion, it gets muted. The deletion of a module can be cancelled by tapping it’s button again. Note, however, that once the “settings” button is released, there is no undoing.

7.1.3.4 creating modules

The creation of a module follows two steps: selecting an empty button (which is not lit), and then selecting the desired new module type throughout the module creation menu.

1. While in super-interactor mode, tap a button in the button matrix which is not coloured (unlit)



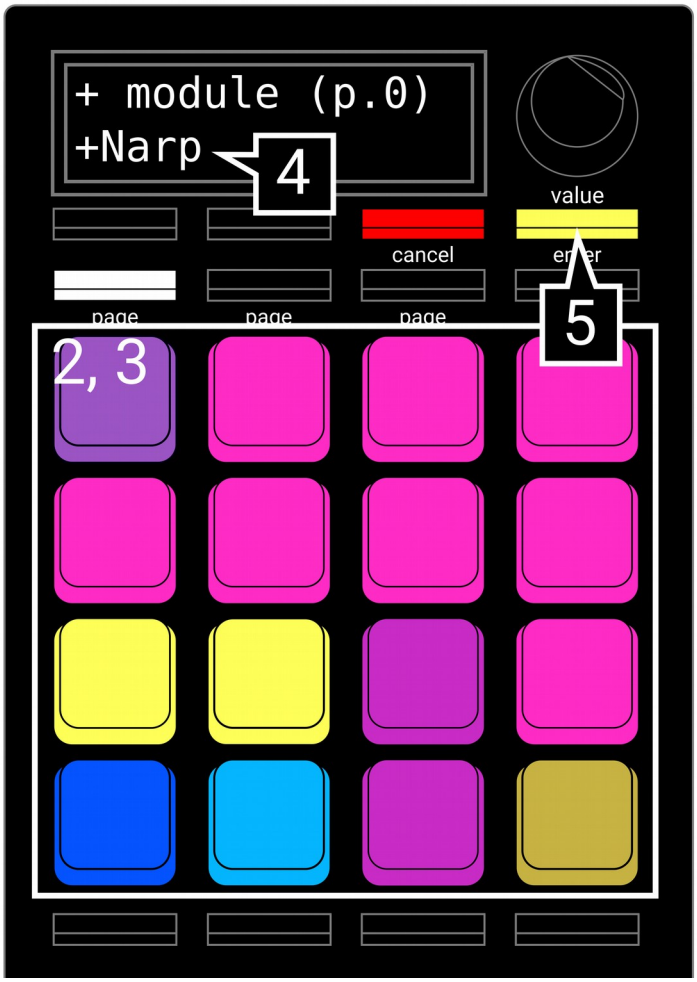
Creating module, step 1

2. The button matrix changes colours. It is now displaying the available modules to create.

3. Tap many buttons in the button matrix. The name of each module will be displayed in the screen. Each module type is represented by a colour as well.

4. Select the magenta button, which is named *narp*.

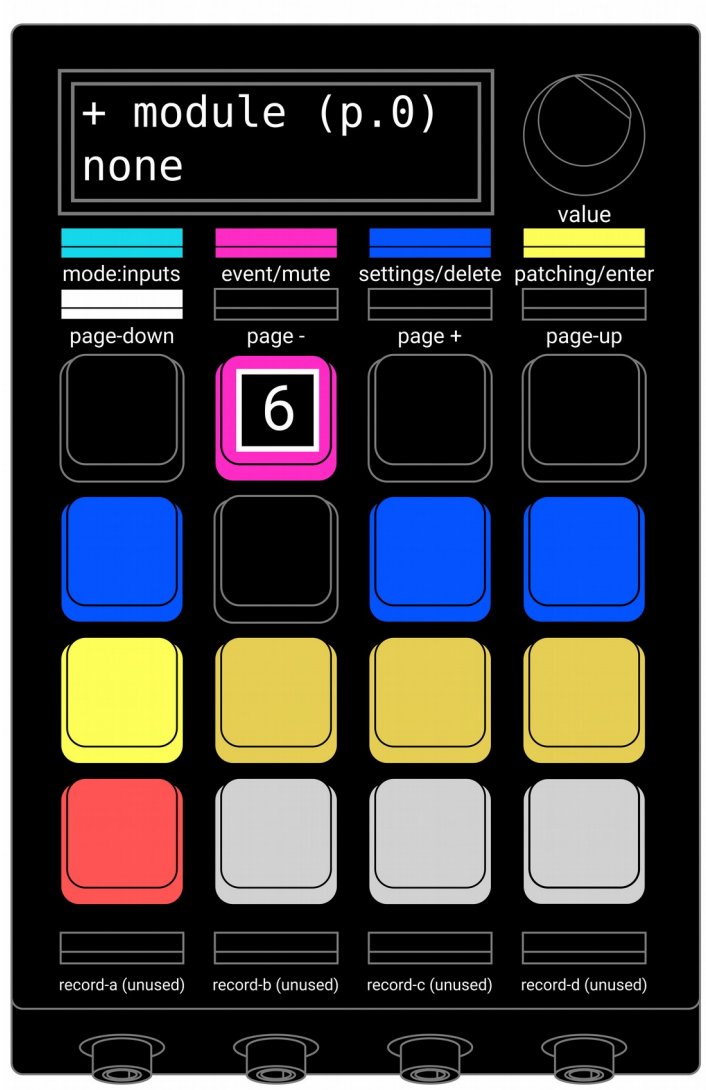
5. Press the "patching" button



Creating module, steps 4-5

Tip: inside the module creation mode, it is possible to exit without creating a new module by pressing the "settings" button. This closes the menu, and goes into super-interactor mode again.

6. Calculeitor goes back to *super-interactor* mode, and the new module appears in the matrix button that you pressed initially.



Creating module, step 6

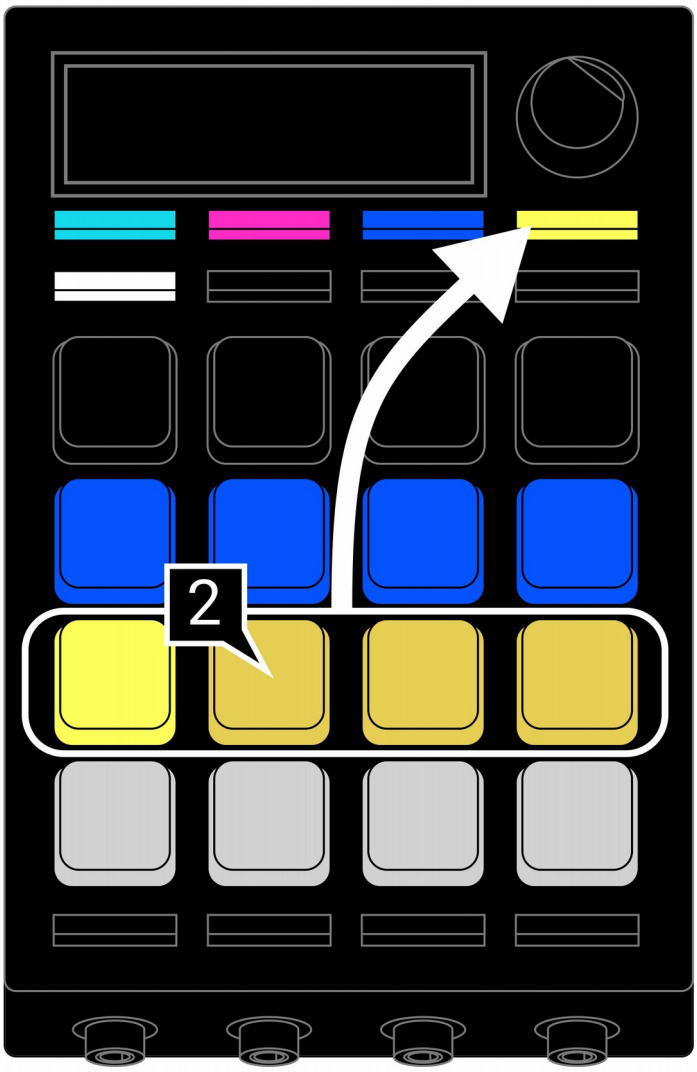
Extra points: connect the new narp module to any of the modules that are displayed yellow. Then, enter into the narp and try pressing some buttons in the matrix! Then try changing the output of that narp to different modules.

Tip: if there are more than 16 modules available, the creation menu has pages which can be explored by pressing the “page” buttons.

7.2 Usage tutorial: Your first performance

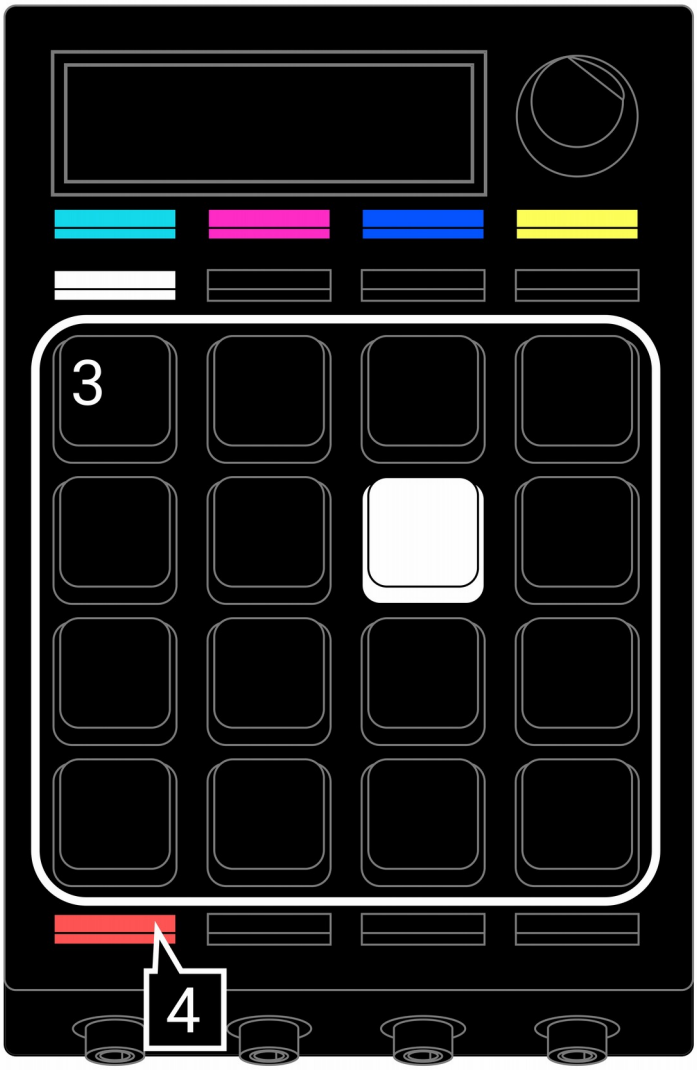
1. Run the Polimod Virtual-Modular environment, and the synthesizer(s) of your choice.

2. Once the super-interactor is displayed, press one of the yellow-coloured modules, and open it



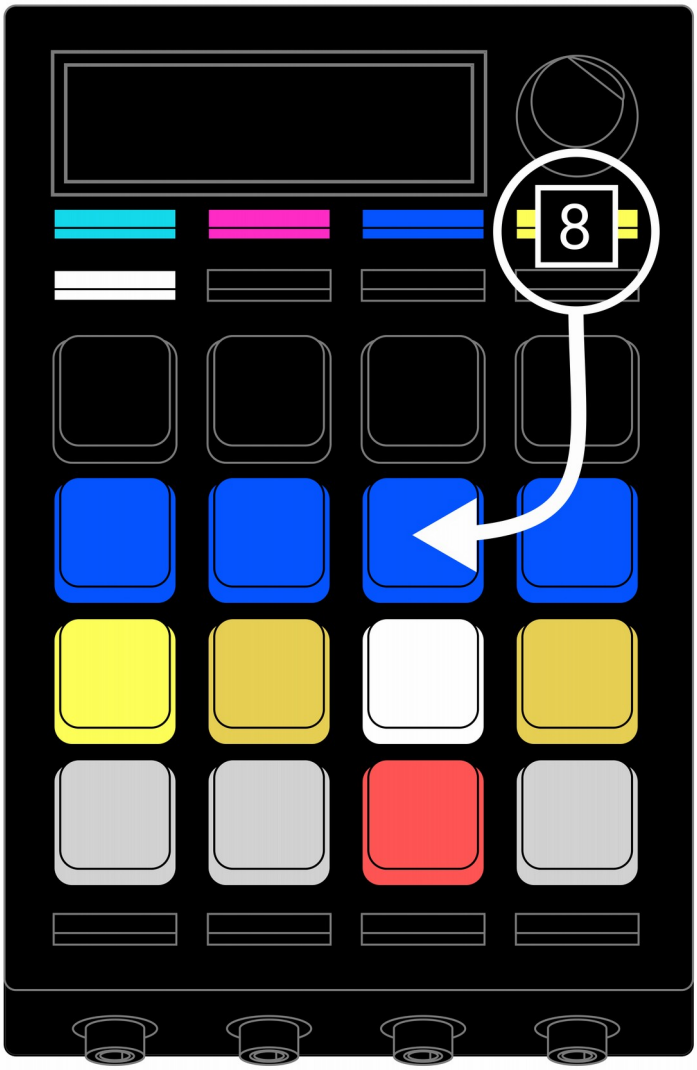
First performance, step 2

3. Play sounds by pressing the buttons.
    - If there is no sound, it means that your chosen synthesizer might have no sound assigned to one of the channels. In this case, go back to the super-interactor, and repeat the step 3.
4. Press the record-a button. It should turn red
    - If it does not turn red, it might be because no module is connected to the current module. In the default patch, all the yellow modules are supposed to have one module connected to them. Simply close the Virtual-Modular environment (use control+c key combination in the command window), and open it again.



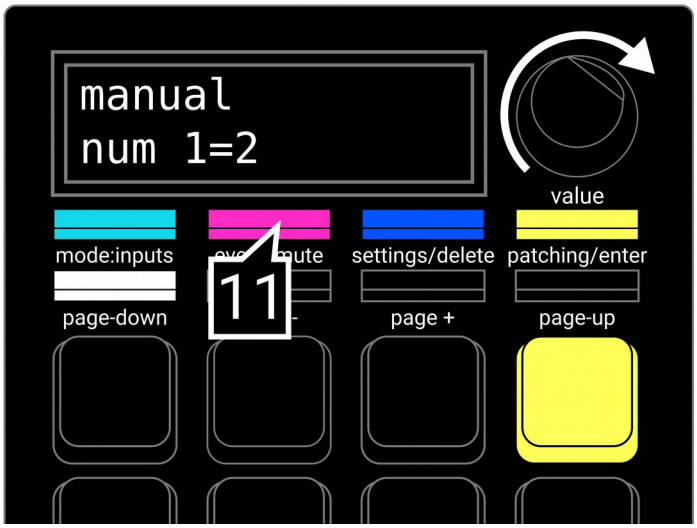
First performance, steps 3-4

5. Play a short pattern, the default tempo is 120 bpm.
6. Press the red button right after you finished playing the pattern.
7. The pattern you played should repeat, with a quantization applied to it.
8. Press the “patching” button, and enter into the module coloured blue which is right above the module that you just selected (the last selected module should appear white in this context).



First performance, step 8

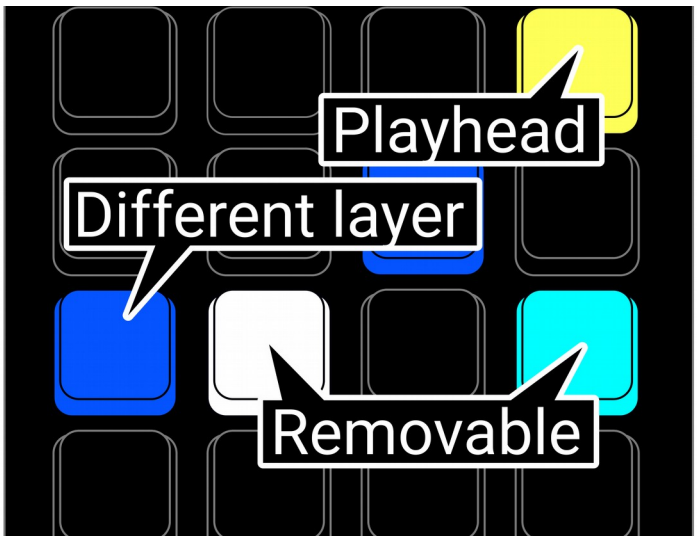
9. You should be able to see the sequence that you just played.
10. Modify the sequence: press different buttons in the button matrix, to program events
11. Select different notes: press the “event” button (second button in the top) and rotate the encoder, to select different notes or sounds. Keep the number below 16 for drums, and do not use negative numbers.



First performance, step 11

The sequencer works by layers, allowing you to produce polyphonic compositions. Each layer is one note (it can be a bit more complex than that, but let's leave it for later). Rotating the encoder changes your point of view from one layer to another.

Tip: to remove an event, you need to be in the same *layer*. Events that are removable appear white or cyan (greener blue)



First performance, colour symbology

12. Repeat this operation as many times as you want. You can combine this tutorial with the concepts explained in the "calculeitor interface introduction/super-interactor".
    - Try creating new sequences and removing older ones.
    - Try connecting the sequencers to other modules.

Are you able to discover your own ways of performing already? Annotate your discoveries so that you can replicate them!

7.3 Usage manual: event configurator

The event configurator is used in most modules. It is used to select a message, which will presumably be sent to the module's output. Think of it as a word selector: it select what the module will tell the other modules.

An event-message is composed by many numbers. With the event configurator, it is possible to select each of these individual numbers.

7.3.1 Pre-configured events

While you hold the *event configurator*, some matrix buttons appear blue, and some other appear magenta. The blue buttons are used to select pre-configured events (for easier use). By tapping these blue buttons, different types of events are configured for you. The most used is the first one, named *note trigger*. The *note trigger* events are those whose first number is 1.

The last blue button, named *manual*, lets you manually configure each of the four numbers of the event.

7.3.2 About events

The effect that an event-message has over a module depends on the numbers that it contains. The most important number is the first one (also named num[0], or head); this one determines the function type of the event. Some examples of function type are *clock beat*, *trigger note* and *change rate*. It is analogous to the first nibble of the MIDI header byte. The other numbers usually give more details about that action. For example, if the event is of *note* type, the following numbers may determine the note number (pitch, or timbre), channel, velocity, among other things.

Knowing the role of each number depending on the header can be a bit tricky to remember. This is why pre-configured events may be useful: if you select a clock event, the following numbers will be renamed as *cycle* and *micro step* accordingly. This makes it easier to choose an event. Try different things! If you were strongly expecting something to happen, but it doesn't, it could mean that you have a feature request, specially once you become well acquainted with Polimod.

There is a known bug where a pre-configured event gets a header that doesn't correspond. We haven't figured out yet what causes this.

7.4 Usage manual: Sequencer

7.4.1 Recording

In default mode, a sequencer tries to adjust the length of the sequence to the performed pattern, without leaving silent gaps. This means that most of the times, parts of the recording are cut off the sequence.

- These events can be recovered using “shift + compensate” technique
- The recording mode can be changed so that the length does not change
- By default, the sequencer goes to “overdub” mode right after recording.

You can change the recording behaviour by using the *recording configurators*. These are present in the last buttons of the settings menu.

7.4.2 Creating and removing events

Events are created simply by pressing a button in the matrix. If an event is present in the same button as the pressed, and this event has similar characteristics, it will be removed.

The duration of the events is equivalent to the amount of time that the button is pressed when creating it.

To remove events regardless of the layer, press the *shift* button (first one, top row). Events of different header, however, can not be accessed using this technique.

From practice, it becomes clear that it is better to dedicate sequencers to specific functions. This makes the navigation and edition a lot easier. If there are too many different events (e.g. clocks, and notes in the same sequencer, or many different voices and instruments) in a sequencer, it becomes hard to remove specific events.

7.4.3 Choosing the event / layer

In this sequencer, layer is equal to the event being created. The underlying assumption is that you will rarely need to create two events of similar characteristics at the same time.

To select the event, press the *event button*; this displays an *event configurator*. The event’s first number (note, if it is a trigger event) becomes the layer. The numbers 2 and 3 are ignored with respect to the layer. The events whose number 2 is different than focus, however, appear in cyan instead of white.

By pressing the *shift* button (first, top row) you can set the focus to every event with the same header. Events with other headers appear red. This is useful to remove events

without having to meticulously go layer by layer. To remove events of different headers, however, there is no shortcut.

When an event is removed using shift, the *event configurator* is adjusted automatically to be the same as the removed event. This is handy for when you need to move an event to a different place.

7.4.4 Changing the length

The sequencer offers many different ways to change the length, because changing length can be a way to perform. All the length configurators are present in the configuration menu; *settings button*, third button in the top row.

7.4.4.1 Traditional length adjustment

- While holding the settings button, press matrix button 0.
- The screen should read “set loop length” and " to 16"
- You can release the buttons.
- Rotate the encoder. This will change the loop length value. The effect will become very clear if you make the sequence shorter than 8.

7.4.4.2 Folding

Folding is changing the sequence length to the double or half of the current length. It will be easiest if you set the sequencer length to 8 or 16, to understand the effect of folding.

7.4.4.3 non-destructive folding

- while holding the settings button, press the matrix button 1
- The screen should read “set fold” plus something like “2^4>16”

  among the numbers 2^4>16, the first number represents the folding base, and the second number the exponent, and the third number is the current loop length. If you wish to use other base, such as 3 (meaning that the length triplicates instead of duplicates), hold the shift button while you rotate the encoder[22]

- Rotate the encoder. The loop length changes drastically to halves or doubles of the current length.
- The length can be re-established, and the sequence remains. This means that you can use non-destructive folding to hide patterns that can appear later.

---

22 not implemented yet.

7.4.4.4 destructive folding (folding!)

- while holding the settings button, press the matrix button 2
- The screen should read “set fold!” plus something like “2^4>16”
- Rotate the encoder up. This duplicates the sequence length, but the new sequence instead of being blank, it is a copy if the first half of the sequence.
- Rotate the encoder down. The sequence out of the range is not only hidden, it is also cleared.

> Needless to say; choosing the wrong type of folding can be fatal to a performance. Always put attention to whether the action contains the “!” character. This character indicates that the folding is destructive.

7.4.5 Paging

If the length of the sequence is larger than 16, it is possible to edit the entire sequence by using the page buttons. For lengths higher than 128 it gets harder to navigate.

7.4.5.1 Page buttons

The second row of buttons are used to jump into the pages 0 to 3 (steps 0 to 63). The buttons underneath the matrix buttons are used for the pages 4 to 7 (steps 64 to 127).

Any page can be selected by using the *page configurator*. This configurator is selected by pressing the matrix button 5 while holding the *settings* button. The screen should read “set page”, followed by the current page.

7.4.6 Shifting the sequence

By shifting the sequence, two different things can be understood, both of which are possible:

- changing the position of the play-head, which changes the coordination of the sequence with respect to any other sequence
- changing the position of the sequence within the loop, without affecting its coordination to other sequences (the sequence is actually shifted, but the play-head is shifted too, to compensate).

7.4.6.1 Compensated shift

- While holding the settings button, press the matrix button 3
- The screen should read “set shift+cpte.” and “to 0”. This stands for “shift and compensate play-head position”

- Rotate the encoder either side, explore, for example going down from –8 and then up to +8. Observe what happens to the sequence.

Compensated shift is specially useful to put the events where you expect. A typical situation is that, after recording, say, a drum pattern, the strong notes end up in odd buttons. It is often expected in a sequencer that the strong events are placed in even buttons, such as the button 0.

Another use to this shift, is to hide parts of the sequence. Let's say that you want to smoothly make a melody to disappear. You can slowly shift the melody outside of the sequencer boundaries,

7.4.6.2 play-head shift

play-head shift merely adds or subtracts to the play-head position, causing the sequence to offset from it's original position.

- While holding the settings button, press the matrix button n¤7 (it is right over the compensated shift button!)
- The screen should read "set drift substep"
- Rotate the encoder left or right. The sequencer changes its position

This technique is specially useful when you have the sequencer well synced to other sequencer or gear, but the step position is not right. First determine how many steps you need to offset it with respect to the other sequences, and then rotate the encoder for the same amount of clicks that you calculated.

Other ways to offset the sequencer are by external module or by jumping. When a sequencer receives a trigger on (or note) event, it jumps to the step indicated by the number [1]. It is possible to manually jump to a step by pressing *shift*+*event*+the desired step button.

7.4.7 Sequencer rate

A sequencer maps directly one step per step. It is possible, to make it faster or slower (e.g. double or half the speed). This affects the amount of clocks that it takes to advance one step; and the length of those events too (for the case of trigger events).

- While holding the "settings" button, press the matrix button number 6 (third button in the second row.)
- The screen should read "set step length" and "to 1"
- Rotate the encoder. This effectively causes the sequence to run at different speeds.

This technique is specially useful for events that need to happen less often. Let's say that

you have one sequencer with length 5, while all your song is playing at 4/4 metric. In this case, you can add a sequencer that resets the position of the 7 steps sequence back to 0 every 128 steps. Another example is when you are using the sequencer to set the tonality of a melody at every repetition of the melody.

7.5 Description of various modules in the Virtual-Modular environment

In devices that have any type of sequence, there are three different step measures, that are related. A step, is the musical step that we are used to, in a sequencer, the step corresponds to the position of the playing head. A sub-step is a translation between steps from a clock sync source, and the sequencer's step. The reason for having a sub-step, is to allow the musician to have slower sequences that wait, for example, eight steps from the clock to advance just one step in the sequencer. If the example sequencer step rate is set to 1, then steps are equivalent to sub-steps. The smallest clock measure so far are the micro-steps. The idea of micro-steps are taken from the MIDI specification, where 24 clock sync signals are specified to conform one quarter note ("Summary of MIDI Messages" 2018).

Those modules whose function principle is simple are presented with a code which explains the basic working principle. The code used in the actual prototype is more complex because it needs to be secure against failure and interact with a user interface. For every case the module also implements more features which enhance the versatility of the module.

A function mapping of inputs is also provided. In the context of the context of Virtual-Modular environment only two inputs were possible per module given the interface that was used. These are provided in a list, for each input (main and recording inputs), a list of headers are provided and what effect does each header produce in the module. As it was described, the event-messages are defined as variable duration, consecutive numbers. The header being the first number, and the following numbers being named consecutively. According to this the event-message is described as `[header, number 1, number 2, ... , number n ]`. Some of the functions that are described in this list may have not been yet implemented at the time this document was printed.

7.5.1 Preset-kit

Minimal procedure (expressed in javascript):

```
Module=function (environment){
    var self=this;
    var kit=Array (16);
    this.onMessageReceived=function (message){
```

```
        if (message[0]==headers.triggerOn){
            if (kit[message[1]]){
                if (kit[message[1]].active) self.sendMessage (kit[message[1]]);
            }
        }
    }
    function setPreset (number,event){
        kit[number]=event;
    }
    function mutePreset (number){
        kit[number].active=false;
    }
    function unmutePreset (number){
        kit[number].active=false;
    }
}
```

A preset-kit offers a fast way to map a set of 16 event-messages to other 16 event-messages. This make it possible to remap the outcome of a sequencer without having to edit the sequence step by step. It also allows to filter events of a sequence by muting or unmuting presets.

- Inputs:
  - Main:
    - **Trigger on**: the preset numbered with the event-message number 1 is triggered to the output. All the numbers present in the incoming message, but not defined in the preset are copied to the output as well (this provides the possibility to have dynamic velocities on synthesizer-triggering messages, for example)
  - Recording:
    - **Record default**: the recording header is removed from the message, and all the subsequent numbers are shifted left.[23] The resulting event-message is assigned to a preset number. The preset number to change upon recording message is consecutive, meaning that they are recorded consecutively. If the last preset is reached, this count starts from 0.

7.5.2 Harmonizer

Representation:

Minimal procedure (expressed in javascript):

```
Module=function (environment){
    var self=this;
    var scale=[0,2,4,5,7,9,11];
    this.onMessageReceived=function (message){
        if (message[0]==headers.triggerOn){
            var noteIn=message[1];
```

23 in most programming languages there is a `shift ()` function which does exactly this.

```
            var octave = Math.floor (noteIn / scale.length);
            var grade = scale[noteIn % scale.length];
            var noteOut = grade + (12 * octave);
            self.sendMessage ([message[0],noteOut]);
        }
    }
}
```

General:

Harmonizer maps inputs into outputs that belong to a musical scale, thus creating an abstraction of harmony. A musical scale consists on a subset of event-messages out of a 12 note chromatic scale. From an incoming trigger event-message, an octave number and grade number are extracted by using `floor (number[1]/scale.length)` and `number[1]%scale.length` respectively. These two factors are used to translate the incoming number 1 into a scale grade[24] and an octave number. The scale grade is selected from the scale array, and added to the extracted octave times 12.

In this mode of operation, the output range of the incoming stream of events is expanded. This is because the number of selected output notes can only be the same length or smaller than 12, which is the times the extracted octave is multiplied by. This has proven to be problematic in some specific scenarios. For instance, changing the scale to a newer scale with smaller amount of grades could cause the resulting pitches or numbers to change their range drastically. To solve this problem, in the described mode of operation there needs to be a modulation centre note. This one determines which note does not get transformed. The notes lower to this pivot notes get lower than the input, and the notes higher to this pivot note get higher. This is a process similar to scaling in the ambit of graphics. The selection of a pivot note in this case is analogue to the selection of a centre in a scaling operation.

An alternative mode of operation for this module, which is not prone to drastic range changes is to round the incoming notes into grades instead of expanding them. For each incoming note, the octave and grade number are extracted. Instead expanding the range of the incoming note by mapping it into an array, this mode intends to keep the range and *round* the incoming note to the nearest grade. For this, the octave and grade extraction functions are respectively `floor (number[1]/12)` and `number[1]%12`. Among the scale array, the number is sought which as the smallest difference (higher than 0) from the extracted grade number. This found number is used as the output grade number, to which the octave times 12 is added.

The harmonizer has 16 memories for scales, that can be configured freely to any possible scale within the western 12 chromatic notes system. This allows the fast toggling between different pre-set scales or chords. This allows many interesting modulations; for instance, if the harmonizer is transforming the output of a short musical sequence, this sequence can be modulated along each repetition to form different structures, obtaining a

24 grade is defined here as the number of the note among the subset of notes in a scale scale rather than chromatic note (e.g., note 2 is C# in chromatic, but D in C major).

modulated melody sequence. It is also possible to use a harmonizer to obtain unexpected mapping from patterns of percussion.

A harmonizer features a keyboard style interface. The recording output of a harmonizer consists on the keyboard notes that are pressed i.e. the grades. This allows, as explained, to re-map the recorded sequence into another harmony. The switching of scales is also recorded.

A harmonizer needs to have two event-message configuration layers: one layer of configuration edits the messages that are sent to the output; these overwrite the information coming from the input. The second layer contains the notes that are used in the keyboard, and thus recorded.

- Inputs:
  - Main:
    - **Trigger on** triggers a note on. The second number is remapped to belong to the chosen scale.
    - **Preset change** changes the current scale, effectively changing the way how the incoming notes are remapped to grades. The second number determines the new scale to use.
    - **Rate change** changes the base note, effectively transposing the output chromatically according to the number 1.
  - Recording:
    - **Record default** The recording event-message is shifted to remove the header. The resulting event-message is used as the output operation of the harmonizer.
    - other recording events will be designated in the future to activate or deactivate grades in the scale, and alter different parameters.

7.5.3 Mono-sequencer

minimal procedure (expressed in javascript)

```
Module=function (environment){
    var self=this;
    var pattern=Array ();
    var playhead=0;
    this.onMessageReceived=function (message){
        if (message[0]==headers.clock){
            if (message[1]%message[2]==0){
                if (pattern[playhead]!==undefined){
                    self.sendMessage (pattern[playhead]);
                }
                playhead++;
                playhead%=16;
            }
        }
    }
    function addEvent (step,event){
        pattern[step]=event;
```

```
    }
}
```

General:

Mono-sequencer is a 16 steps sequencer that only allows programming of one event per step, and only allows a maximum of 16 steps of a sequence.

This module is used as testing module to build new versions of the environment. During this development, the environment has been re-programmed 5 times in three different languages, with different levels of success. The mono-sequencer is the best test module to work with, because it produces inputs and outputs, it has a simple functionality to program, and it can be modified easily to become a full sequencer when the environment is completed further.

7.5.4 Sequencer

Sequencer represents a classical style sequencer, with some additional features for better performability. There are many edition tools in the sequencer that do not target a specific musical modulation, but offer generic pattern handling options, allowing unexpected modulations by using combinations of these modifiers. Not too many modulations are possible with the sequencer, however. The broader range of modulations are achieved by using the sequencer in different combinations with other modules.

The sequencer evolved from a mono-sequencer that was used to test the first prototype of the environment, to a sequencer that can hold a wide variety of musical expressions, although always in a quantized format. The first additions that were inspired by Elektron sequencers was the *look sequencing* which consists on establishing an event recurrence that is different from the sequencer length, allowing to program events that recur more than once in a sequence. Having evolved from a mono-sequencer, the sequencer holds a vernacular quantized step memory. There was an immediate realization for need of real-time recording capabilities, which is present on most sequencers. Because of the mentioned quantized memory, the mode of recording is most similar to the Electribe's procedure because of its similar quantization.

Inspired in some of the Maschine affordances to play, the sequencer also acquired the ability to reset position on the real time, to allow performing with polyrhythm in an expressive way or doing jumps in the sequence in the style of cue-point jumping. This led to an interface procedure where tapping a sequencer button would perform the jump. The musical tracks in Maschine are tempo-locked, meaning that there is not much liberty to drift tracks away one from other, but it is also a very comfortable feature for most of the time. This realization, together with the need to automate the mentioned step-jumping, inspired the step-jumping message types in the sequencer, allowing to create sequencers that are strictly tempo-synchronized (if they are being triggered periodically at the same time) and also sequencers which are constantly jumping off of sync (if the sequencers are triggered differently).

- Inputs:
  - Main:
    - **Micro step** the second number indicates the amount of micro-step for each step of this clock, the third number indicates the micro-step number within the indicated micro-steps. When the third number % second number equals zero, one sub-step is advanced.
    - **Trigger on** jumps to the step indicated by the second number and sets the sequencer to play
    - **Trigger off** stops the sequencer playback only if this functionality has been activated by the user.
    - **Rate change** changes the amount of sub-steps per step. By default this value is 1, which makes the sub-steps a synonym of steps. Different values, however allow the sequencer to run at different rates (for example half or double the speed).
  - Recording:
    - **Trigger on** The event-message is added to the sequencer in the current playback position. This facilitates real-time recording of events from modules such as harmonizer or preset-kit.

Based by Maschine's handling of patterns in a *sound*, or similarly Ableton's handling of clips in a *track*, a sequencer should be able to hold multiple sequences that can be exchanged. This allow an additional axis of expression where a sequence can evolve in many ways and still be able to get back to the initial point. There has been a history of speculation about this feature that goes all the way back to the initial idea of this project, even after there was an idea of making a modular environment. The first idea, was to have a generative function that alters any existing sequence to any extent. This would allow to generate many variations of a user-defined sequence by turning a knob, according to a function that guarantees consistency, thus allowing to turn the pattern back to the original state. The second idea consisted in having a multi-clip sequencer, which derived into the creation of the multitape module and has not attained yet a satisfactory state.

Current ideas for this feature comprehend the implementation of a pattern history, similar to the *undo* history of user-friendly computer software, or the selection of looping points, which could be shifted freely to reveal different sections of a longer pattern. Although the *undo* history based pattern-variation procedure seems like the most user friendly and attractive, it poses some questions that are hard to answer; for instance, if a user goes back to *undo* history and makes a change, what happens with all the *redo-able* states? Would the history discard the original pattern of that undo stage, or would it include a new state in the *redo* stack? Would the history become more like a tree, whose different branches could be explored? In such case, what kind of user interface would prevent the user from getting lost? If linear, the ability to go back in history would become like an array of versions each of which can be customized; but in that case, what defines the limit between one version and another? Would, for instance each newly introduced or removed event create a new version, or the user would need to establish by hand the division between versions? All these questions are very open to different answers, and due to time

availability an optimal solution has not yet been decided.

7.5.5 Narp

Narp is a stripped version of an arpeggiator. The button matrix is used to activate or deactivate different events. The event on each button consists on a trigger on message, with a second number defined as the button number plus a base displacement. The narp allows events with numbers only in a range of 16, and only does the operation in order from the lowest to the highest active number. If events are received with a higher number, a remainder operation takes place (% 16) to set that number within range. The idea behind such a limited module, is to foster free and safe exploration. This module is very suitable to produce ever-drifting polyrhythms, since the length of an arpeggiated sequence depends on the amount of notes included in the arpeggio cycle. The other advantage of the narp, is that it restricts the output events into one event. This can be useful to produce an accompanying arpeggio to a melody, using a different MIDI channel in the output.

- Inputs:
  - Main:
    - **Micro step**: the second number indicates the amount of micro-step for each step of this clock, the third number indicates the micro-step number within the indicated micro-steps. When the third number % second number equals zero, one sub-step is advanced.
    - **Trigger on**: activates the arpeggiator note indicated by the number[1]%16 operation.
    - **Trigger off**: deactivates the arpeggiator step which was activated by the note on that possessed similar values.
  - Recording:
    - **Trigger on** Narp records the notes that are activated and deactivated as *trigger on* and *trigger off* events. It is intended to record the changes on the sub-step to step ratio as well, although this feature has not yet been programmed.

7.5.6 Arpeggiator

An arpeggiator is a typical building block in music, and perhaps it is the one that most strongly suggested the need for a modular environment that treats event-messages as signals to be processed through effects. Different from the narp, an Arpeggiator stores the incoming notes in order, and plays them alternatively on each step according to this order. Different arpeggio patterns can be achieved if its clock source is sequenced by an external sequencer. Unlike the narp, the arpeggiator can hold an arpeggio of notes in any range and with many different properties, in an order that is not necessary incremental.

The Arpeggiator can be used either as an effect that interrupts and modifies the stream of event-messages, or as a module that can receive the notes as recording notes, feeding them back to the module that is originating them. This allows an easier mode of use,

because while performing a pattern on a module, the user can activate or deactivate the arpeggiator without having to change the connections but just by enabling or disabling the recording.

One feature which may be implemented is the addition of any incoming note regardless of the header to the memory. The only exception in this case, would be timing messages received in the main input.

- Inputs:
  - Main:
    - **Trigger on** the event is added to the arpeggiating memory
    - **Trigger off** the events whose second and third number are the same, is removed from the memory.
    - **Micro step** a micro step is advanced. This leads to the advancement of steps according the *step ration* user setting. When a steps advance, one consecutive event from the memory is played.
  - Recording:
    - **Trigger on** Recording events in the Virtual-Modular environment, have the same effect as normal events, allowing different patch routes.

7.5.7 Game of life

Game of life was the first module made to consider ideas of more experimental modules, allowing a broader area of musical experimentation based on unexpected behaviour. This idea is obviously borrowed from the more generative Euro-rack modules such as Makenoise's *Maths* or Music Thing Modular's *Touring Machine*, and was closely based on Reaktor's Newscool patch. This module uses the 16 buttons matrix as a grid to run Conway's *game of life* algorithm (Jiameson 2016).[25] The grid was modified in order to "wrap around" the effect of the algorithm, meaning that the first row of cells are affected by the last and vice versa, and the last column of cells are affected by the first column and vice versa.

- Inputs:
  - Main:
    - **Micro step** the second number indicates the amount of micro-step for each step of this clock, the third number indicates the micro-step number within the indicated micro-steps. When the third number % second number equals zero, one sub-step is advanced.
    - **Trigger on** activates the arpeggiator note indicated by the remainder of the second number when divided by 16
    - **Trigger off** deactivates the arpeggiator note indicated in the same way as the *trigger on*.
    - **Rate change** sets the amount of sub-steps that must be counted to

25 At each step of this module, each cell that is "living", will produce a [trigger on] event whose second number equals the grid button number plus a global displace value (Jiameson 2016).

advance one step
        - **Rate change 2** how many sub-steps a note should be held on once it is triggered by the arpeggiator. This allows for fractions of a sub-step.
    - Recording
        - **Trigger on** The mechanic of note-off and note on in the game of life is the same as in a narp; taking the same effect as if it was received through the main input.
        - **Trigger off**

7.5.8 Clock based delay

The delay stores any input event in a memory except for the clock events, and propagates them to the output once the user-specified delay time is reached. The time is counted in accordance with the received clock events. It is possible to build a feedback mechanism to the delay using operators. However, this was integrated into the delay module itself in order to simplify this common procedure. The use of constructed feedback remains interesting because it allows chaining effects which could produce unusual results.

- Inputs:
    - Main: any incoming event except for clock and rate change events are stored in a memory.
        - **Micro step** the micro step is advanced, which can result in the triggering of events which are stored on memory, depending on the delay time user setting. All the events which are propagated to the output are removed from the memory.
        - **Rate change** changes the delay time setting.

7.5.9 Route-sequencer

The route-sequencer forwards all the incoming events, except for the clock events, into one of its outputs. The output to which the events are forwarded are determined by a step sequencer. The step sequencer advances in position in relation to the received clock. The rate of this sequence is determined by the clock ratio specified by the user.

7.5.10 Chord generator

The chord generator module produces a simple transformation exclusively to trigger on events. Each received trigger event is treated as a note, and many replicas of the initial note may be generated with different values on the number 1. The amount of times to replicate the event and the value of each replica relative[26] to the input event. The relative value of each replica is determined by the user through a simple matrix interface. The

26 meaning that the output value is equal to the input value plus the corresponding number.

interface represents a *pivot* note as a red square. The active state of each square in the matrix is toggled by pressing. Active matrix buttons account for one copy of the original event, whose relative value is represented by its distance from the representation of the root. The distance is not measured spatially, but sequentially in a way similar to the occidental flow of text; meaning that an active square immediately below the root is not at distance 1, but at distance −4. The different copies of the root event, henceforth can be defined by the user as copies below or as copies above the original note.

7.5.11 Operator

Operator is the implementation of one of the basic elements discovered at the end of the *buildification* process described earlier as its function was often speculated would be useful. An operator simply changes an input event-message by applying a mathematical operation to each one of the message's numbers; hence its user interface consists on a set of pairs of operations and numbers.

```
[in] ?! [op] notch filter: every event-messages whose [n] number equals to the
operation number is discarded
[in] ? [op] band filter: every event-message whose [n] number differs to the
operation number is discarded
[in] > [op] high pass filter: every event-message is discarded, except if their
[n] number is higher than the operation number.
[in] < [op] low pass filter: every event-message is discarded, except if their
[n] number is lower than the operation number.
[in] = [op] set (or assign): every event-message's [n] number is set to the
operation number
[in] + [op] add
[in] - [op] subtraction
[in] / [op] division
[in] % [op] remainder
```

- Inputs:
  - Main: for each number of the received event, the corresponding operation is performed and fed to the output.
  - Recording An operator can receive recording messages. The recorded message replaces the operation numbers correlatively.

7.5.12 IO MIDI

In the context of the Virtual-Modular environment, this was the module used to output the results of the environment into another environment which could sonify the events (e.g., Pure-Data, Super Collider, Maschine). This module transforms the incoming event-messages into MIDI by applying the operation specified in Fig. 43.

7.5.13 Clock generator

A Clock generator module generates a stream of micro-steps. These are used by some

modules to determine the playback of sequences, arpeggios, or any other time related features. This module does not take any input. It could be stipulated that an input could determine the clock speed in a future implementation. In the current virtual environment it is technically problematic to automate a clock change. This is because javascript does not offer a built-in framework for real time interval functions, and a more complex algorithm[27] was built to keep the relation between javascript intervals and the real-time ones. This leads to tempo changes to take effect gradually instead of instantly.

7.5.14 Bouncer

Another module in the family of basic modules is the bouncer, which casts all the incoming messages as recording messages into the output modules. Implemented as a hardware, the bouncer would not exist since any module could be connected to the recording input of any other module. It was inspired by many different hardwares such as Kaoss Pad or Maschine in their capacity to re-sample their own outputs, allowing feedback in the process of modifying an ongoing pattern.

27 the mentioned algorithm measures the difference between the clock events to the time they were supposed to happen in relation to the real-time clock, and times the next iteration with compensation to this difference.

8 Bibliography & references

"Ableton Manual: Using Push." 2018. Ableton. Accessed December 2. https://www.ableton.com/en/manual/using-push/.

Aldunate Infante, Joaquín. 2013a. "Brocs." *Autotel.co*. January. http://autotel.co/portfolio/brocs/.

———. 2013b. "Brocs: Objeto Para Experienciar Sin Conocimiento, La Música de Forma Auto-Télica." Universidad Diego Portales. https://drive.google.com/file/d/0B_0eojMydbGZcVFuVDJkMVRDb2c/view.

———. 2014. "Licog Composer." *Autotel.co*. January 1. http://autotel.co/portfolio/licog-composer/.

"Analog Four Manual." 2018. Elektron. Accessed July 4. https://www.elektron.se/wp-content/uploads/2017/10/Analog-Four-MKII-User-Manual_ENG-2.pdf.

Andean, James, and Alejandro Olarte. 2012. "Sound, Music and Motion: Sound Art and Music in Cross-Disciplinary Improvisation." In *Third International Symposium on Music/Sonic Art: Practices and Theories*. Centre for Music & Technology, Sibelius Academy.

Arar, Raphael, and Ajay Kapur. 2013. "A History of Sequencers: Interfaces for Organizing Pattern-Based Music." In *Proceedings of the Sound and Music Computing Conference 2013, SMC 2013*, 383–88. Academic Press. http://smcnetwork.org/system/files/A%20HISTORY%20OF%20SEQUENCERS%20INTERFACES%20FOR%20ORGANIZING%20PATTERN-BASED%20MUSIC.pdf.

"Blocks: The Instrument That Grows with You." 2018. ROLI Ltd. Accessed June 26. https://roli.com/products/blocks.

Bostock, Mike. 2017. "D3.js - Data-Driven Documents." https://d3js.org/.

Butler, Mark J. 2006. *Unlocking the Groove: Rhythm, Meter and Musical Design in Electronic Dance Music*. Indiana University Press.

"Circuit." 2018. Novation. June 19. https://us.novationmusic.com/circuit/circuit.

"Circuit Mono Station." 2018. Novation. June 19. https://us.novationmusic.com/synths/circuit-mono-station.

"Circuit User Guide." 2017. Focusrite Audio Engineering Limited. https://d2xhy469pqj8rc.cloudfront.net/sites/default/files/novation/downloads/15792/circuit-ug-en-03-v1-6.pdf.

Cornish, David. 2013. "Watch Imogen Heap's Full Wired 2012 Glove Demo and Performance." *Wired.co.uk*. January 11. https://www.wired.co.uk/article/imogen-heap.

Cox, Carl, and Adam Beyer. 2018. *CARL COX B2b ADAM BEYER at Junction 2*. Mixmag. https://www.youtube.com/watch?v=E20vtLY6MIw.

"Creating MIDI Effects." 2018. cycling74. Accessed August 15. https://docs.cycling74.com/max5/vignettes/core/live_midieffects.html.

"Deluge." 2018. Synthstrom Audible. https://synthstrom.com/product/deluge/.

"Documentation | Konva - JavaScript 2d Canvas Library." 2018. https://konvajs.github.io/docs/.

Doepfer, Dieter. 2018. "Technical Details A-100." Accessed December 2. http://www.doepfer.de/a100_man/a100t_e.htm.

Dylan Wray, Daniel. 2013. "'Algorave' Is the Future of Dance Music (If You're a Nerd) - Creating Music with Computer Code." *Vice Magazine*. Vice Magazine. November 25. https://www.vice.com/en_us/article/bn5zz4/algorave-is-the-future-of-dance-music-if-youre-an-html-coder.

Emmerson, Simon. 2007. *Living Electronic Music*. Ashgate. https://ebookcentral-proquest-com.libproxy.aalto.fi/lib/aalto-ebooks/reader.action?docID=429715&query=.

Fantinatto, Robert. 2014. *I Dream of Wires*.

"Field Kit- Electro Acoustic Workstation." 2018. KOMA Elektronik GmbH. Accessed June 26. https://koma-elektronik.com/?product=field-kit.

Foreman, Darren. 2011. *Beardyman: "Unshaved" at the Udderbelly, London (Episode 1)*. DJ Mag. https://www.youtube.com/watch?v=pnl1R2dUiD0.

Frey, Tim, Marius Gelhausen, and Gunter Saake. 2011. "Categorization of Concerns: A Categorical Program Comprehension Model." In *SPLASH Conference Preceedings*. https://ecs.victoria.ac.nz/foswiki/pub/Events/PLATEAU/Program/plateau2011-frey.pdf.

Gibson, James J. 1979. *The Theory of Affordances*. Lawrence Erlbawm Associates. https://monoskop.org/images/c/c6/Gibson_James_J_1977_1979_The_Theory_of_Affordances.pdf.

Goodacre, Liam. 2018. "Context Sequencer." Accessed June 26. https://contextsequencer.wordpress.com/.

Groves, Wesley. 2016. "Intro to Eurorack Part I: Doepfer's Beginnings and Power Supply Basics." *Reverb*. July 8. https://reverb.com/news/intro-to-eurorack-part-I-doepfers-beginnings-and-power-supply-basics.

Guilford, Joy Paul. 1970. “Creativity: Retrospect and Prospect.” *The Journal of Creative Behavior* 4 (3): 149–68.

Heap, Imogen. 2013. *Imogen Heap Performance with Musical Gloves Demo*. WIRED UK. https://www.youtube.com/watch?v=6btFObRRD9k.

Hesmondhalgh, David. 1998. “The British Dance Music Industry: A Case Study of Independent Cultural Production.” *The British Journal of Sociology* 49 (2). Wiley-Blackwell: 234–51. http://www.jstor.org/stable/591311.

Hilgenfeld, Olaf, and Iftah Gabbai. 2017. *SKINNERBOX LIVE 2017 (Ableton, Moog, Eurorack)*. Skinnerbox. https://www.youtube.com/watch?v=mm2xjtE9Iqs.

Howlett, Liam, Ashley Abram, and Yukako Nakajima. 1992. *Wind It up. The Prodigy Experience*. XL Recordings.

Jiameson, Ali. 2016. “Reaktor’s Newscool.” *Zeroes and Ones*. May. http://alijamieson.co.uk/2016/05/reaktors-newscool/.

Kaltenbrunner, Martin, Sergi Jordà, Günter Geiger, and Marcos Alonso. 2006. “The reacTable*: A Collaborative Musical Instrument.” In *Proceedings of the 15th IEEE International Workshops on Enabling Technologies:Infrastructure for Collaborative Enterprises*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.9219&rep=rep1&type=pdf.

Kavanaugh, Philip R., and Tammy L. Anderson. 2008. “Solidarity and Drug Use in the Electronic Dance Music Scene.” *The Sociological Quarterly* 49 (1): 181–208. doi:10.1111/j.1533-8525.2007.00111.x.

Kirn, Peter, ed. 2011. *Keyboard Presents: The Evolution of Electronic Dance Music*. Backbeat Books.

Koskinen, Tommi. 2015. “The UFO Controller Gestural Music Performance.” Aalto University.

Lai, Chi-Hsia, and Koray Tahiroğlu. 2012. “A Design Approach to Engage with Audience with Wearable Musical Instruments: Sound Gloves.” In *Proceedings of New Interfaces for Music Expression (NIME)*.

Laurel, Brenda. 2003. *Design Research. Design Research: Methods and Perspectives*. The MIT Press.

Linn, Roger. 2018. “Past Products Museum.” Accessed March 5. http://www.rogerlinndesign.com/past-products-museum.html.

Lynch, Will. 2017. “Electronic Artists Should Make Their Own Music.” *Resident Advisor: Opinion*. Resident advisor. Spring 12. https://www.residentadvisor.net/features/2033.

Malbon, Ben. 2002. *Clubbing: Dancing, Ectasy, Vitality*. Routledge.

Maraš, Svetlana. 2011. “Embodied Composition: Treatment and Meaning of Physical Object in Experimental Music and Sound Art.” Aalto University.

https://aalto.finna.fi/Record/aaltodoc.123456789_3591.

Maturana, Humberto, and Francisco Varela. 1980. *Autopoiesis and Cognition: The Realization of the Living*. D. Reidel Publishing Company.

———. 1994. *Autopoiesis: La Organización de Lo Vivo*. Editorial Universitaria.

McLean, Alex. 2014. "Making Programming Languages to Dance to: Live Coding with Tidal." In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design*. Interdisciplinary Centre for Scientific Research in Music, University of Leeds. doi:10.1145/2633638.2633647.

"Merriam-Webster Dictionary, Definition of Divergent." 2018. *Merriam-Webster Dictionary*. Accessed January 22. https://www.merriam-webster.com/dictionary/divergent.

"MIDI Effect Tools." 2018. cycling74. Accessed August 15. https://docs.cycling74.com/max7/vignettes/live_midiexamples.

"New Spikes Milk Edition." 2018. Error Instruments. Accessed June 26. https://www.errorinstruments.com/a-48850247/welcome/new-spikes-white-milk-edition/.

Newton-Dunn, Henry, Hiroaki H Nakano, and James Gibson. 2003. "Block Jam: A Tangible Interface for Interactive Music." In *Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME-03)*, 170–77. NIME.

Niinimäki, Matti, and Koray Tahiroğlu. 2012. "AHNE : A Novel Interface for Spatial Interaction." In *Proceedings of CHI '12 Extended Abstracts on Human Factors in Computing Systems*. ACM.

"Nsynth Super." 2018. google AI. Accessed August 15. https://nsynthsuper.withgoogle.com/.

Octave One Boiler Room Moscow Live Set. 2014. Boiler Room. https://www.youtube.com/watch?v=XW6lxLUBu64.

Parkinson, Adam, and Koray Tahiroğlu. 2013. "Composing Social Interactions for an Interactive-Spatial Performance System." In *Proceedings of the Sound and Music Computing Conference*.

Pettit, Benjamin, Kevin Ford, and Pascal Redpath. 2000. *Super Sharp Shooter. Jungle Classics*. Ministry Of Sound.

Pinch, Trevor, and Frank Trocco. 1988. "The Social Construction of the Early Electronic Music Synthesizer." *Icon* 4 (4). International Committee for the History of Technology: 8–31. http://www.jstor.org/stable/23785956.

Puckette, Miller. 2006. *The Theory and Technique of Electronic Music*. World Scientific Publishing Co. Pte. Ltd. http://msp.ucsd.edu/techniques/v0.11/book.pdf.

"Reactable." 2018. Reactable Systems SL. Accessed August 15. http://reactable.com/.

Reynolds, Simon. 1999. *Generation Extasi: Into the World of Techno and Rave Culture*. First.

Routledge.

Rietveld, Hilegonda. 1995. “Pure Bliss: Intertextuality in House Music.” http://www.snarl.org/youth/purebliss.pdf.

———. 2013. “Introduction to DJ Culture in the Mix.”

“RPG.” 2018. Noise Reap. Accessed June 26. http://noisereap.com/?product=rpg.

Runco, Mark A. 2011. “Divergent Thinking.” In *Encyclopedia of Creativity*, 400–403. Academic Press.

Sánchez Carranco, Camilo. Letter. 2018. “Informal Conversation After Calculeitor Party,” May 4.

“Squarp Pyramid 64-Track Sequencer.” 2016. Squarp Instruments. March 16. http://squarp.net/pyramid.

“Squarp Pyramid Sequencer User Guide.” 2016. Squarp Instruments. August 26. http://www.squarp.net/06_pic_overview/MANUAL/PYRAMID_SEQUENCER_USER_GUIDE_87.pdf.

Straw, Will. 1993. “The Booth, the Floor and the Wall Dance Music and the Fear of Falling.” *The Ethics of Enactment*, no. 8. Public: 169–83.

Sullivan, Paul. 2013. *Remixology: Tracing the Dub Diaspora. Remixology: Tracing the Dub Diaspora*. Reaktion Books.

“Summary of MIDI Messages.” 2018. midi.org. Accessed August 6. https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message.

Tahiroğlu, Koray, Nuno Correia, and Miguel Espada. 2013. “PESI Extended System: In Space, on Body, with 3 Musicians.” In *Proceedings of New Interfaces for Music Expression (NIME)*. Daejeon + Seoul, Korea Republic.

Vahid, Frank. 2007. *Digital Design*. Wiley cop.

van den Oord, Aäron, Sander Dieleman, and Heiga Zen. 2016. DeepMind. September 8. https://deepmind.com/blog/wavenet-generative-model-raw-audio/.

Vasquez, Juan Carlos. 2016. “Defragmenting Beethoven: Sound Appropriation as Bridge Between Classical Tradition and Electroacoustic Music.” Aalto University.

Wahab Lafta, Abdul, and Andre Williams. 2010. *Original Nuttah. Jungle Classics*. Ministry Of Sound.

Walsh, Michael. 2018. “12 DJ Tips + Techniques to Improve Your Live Sets.” *Dubspot Blog*. February 8. http://blog.dubspot.com/dj-tips-techniques-to-improve-your-live-sets-performances/.

Warner, Daniel. 2017. *Live Wires*. Reaktion Books.

Waters, Simon. 2007. “Performance Ecosystems: Ecological Approaches to Musical

Interaction." *EMS: Electroacoustic Music Studies Network*. http://www.ems-network.org/IMG/pdf_WatersEMS07.pdf.

Watts, Reggie. 2013. *EHX Reggie Watts Explores the New 45 000 Multi-Track Looping Recorder*. DJ Mag. https://www.youtube.com/watch?v=0gKWfvd-chA.

White, Dan. 2018. "OWOW Midi Controllers: Small + Simple Design, Advanced Controls." Dj Techtools. June 8. http://djtechtools.com/2015/06/06/owow-MIDI-controllers-small-simple-design-advanced-controls/.

Witts, Richard, and Karlheinz Stockhausen. 1995. "Stockhausen Meets the Technocrats." *The Wire Magazine*, November.

You, Hsiao-chen, and Kuohsiang Chen. 2007. "The Theory of Affordances" 28 (1). Elvesier. https://www.sciencedirect.com/science/article/pii/S0 142 694X06 000 494.